electronics

MDPI

Article

Designing and Evaluating a Flexible and Scalable HTTP Honeypot Platform: Architecture, Implementation, and Applications

Matej Rabzelj [1,*], Leon Štefanić Južnič [1], Mojca Volk [1], Andrej Kos [1], Matej Kren [2] and Urban Sedlar [1]

[1] Faculty of Electrical Engineering, University of Ljubljana, 1000 Ljubljana, Slovenia; leon.stefanic-juznic@fe.uni-lj.si (L.Š.J.); mojca.volk@fe.uni-lj.si (M.V.); andrej.kos@fe.uni-lj.si (A.K.); urban.sedlar@fe.uni-lj.si (U.S.)
[2] Aviat Networks, 1236 Trzin, Slovenia; matej.kren@aviatnet.com
* Correspondence: matej.rabzelj@fe.uni-lj.si

Abstract: Digitalization of our economy and society has ushered in notable productivity increases but has also exposed more of our infrastructures and systems to cyberattacks. This trend is exacerbated by the proliferation of poorly designed Internet of Things (IoT) devices and cloud services, which often lack appropriate security measures, either due to bugs or configuration mistakes. In this article, we propose, validate, and critically evaluate a flexible honeypot system based on the Hypertext Transfer Protocol (HTTP) that can mimic any HTTP-based service and application. This covers a large share of IoT devices, including black box devices with no software or firmware available for emulation, as well as cloud- and web-based services. We validate the system by implementing 14 services and by running a 4-month experiment, collecting data from attackers. We propose a novel data enrichment mechanism for identifying internet scanning services, as well as several other data collection and enrichment approaches. Finally, we present some results and visualizations of the data collection experiment, demonstrating possible applications and future use cases, as well as potential drawbacks of such systems.

Keywords: cybersecurity; honeypot; honeynet; HTTP protocol; Internet of Things; cloud computing; fingerprinting; data fusion

Citation: Rabzelj, M.; Južnič, L.Š.; Volk, M.; Kos, A.; Kren, M.; Sedlar, U. Designing and Evaluating a Flexible and Scalable HTTP Honeypot Platform: Architecture, Implementation, and Applications. *Electronics* **2023**, *12*, 3480. https://doi.org/10.3390/electronics12163480

Academic Editor: Elias Stathatos

Received: 31 July 2023
Revised: 14 August 2023
Accepted: 14 August 2023
Published: 17 August 2023

1. Introduction

The rapid development of modern technology has enabled the digital transformation of the economy and society, as well as the digitalization of domains that provide the underlying environment for the stable and secure functioning of nations and states as a whole. This includes essential services, such as energy, health, transport, water, banking, communications, and also defense. Billions of Internet of Things (IoT) devices are being used today in safety-critical applications, such as industrial control systems (ICSs), eHealth, critical infrastructures, public protection and disaster relief (PPDR), vehicular ad hoc networks (VANETs), and others [1,2]. While representing essential infrastructures that create massive volumes of data, drive industries, and create financial opportunities, the domain has accordingly attracted the significant attention of cybercriminals and today represents one of the most complex and lucrative cybersecurity targets. IoT and cloud computing, in fact, represent two of the most relevant scenarios for cybersecurity, the application and management of which are also the main barriers for its extended deployment and adoption in different verticals [2–4]. In response, cybersecurity in both domains has seen significant developments in recent years. While cryptography, firewalls, anti-malware and anti-virus solutions, intrusion detection systems (IDSs), and intrusion prevention systems (IPSs) play an important role in protecting such systems, specific deception technologies are also

needed, i.e., honeypots and honeynets, designed to deliver transparent observations of cyberattack behavior and collection of more detailed and actionable intelligence [2].

The use of honeypots in information technology (IT) environments has served for decades to detect and monitor cybersecurity trends. Some of the most well-known and established general frameworks include a low-interaction honeypot for attack and malware detection, Dionaea [5], SSH/Telnet honeypot Cowrie [6], and Honeytrap solution [7,8]. In IoT specifically, however, the use of honeypots and deception technology continues to be associated with numerous challenges that originate from the fact that we are faced with an extremely heterogeneous and highly distributed ecosystem of devices not seen in any other domain. The complexity of tackling vulnerabilities of existing and newly deployed IoT devices and cyberattack trends by means of using honeypots is extreme and determined by the volume and diversity of devices that must be mimicked, both in terms of supported services and in their physical connectivity properties [1,9]. The range of risks is immense, and so is the range of types of cyberattacks. Some of the most common examples of IoT characteristics creating vulnerabilities are dynamic and transient device connectivity, weak authentication protocols, physical access to devices in unattended premises, unprotected wireless networks connecting the devices into the system that allow for eavesdropping, limited availability of storage, energy, and computational resources preventing installation of more complex security protection, billions of deployed IoT devices that are left unattended or cannot be upgraded with appropriate security mechanisms, as well as the fact that the vulnerabilities of devices extend into the connected services and applications, which makes wide-scaled attacks specifically challenging [1,2,9,10]. Moreover, as many IoT devices heavily rely on the web service architecture and make use of web technologies and protocols, such as the Hypertext Transfer Protocol (HTTP), to facilitate remote management, configuration, and cloud communication, this also exposes them to an immense number of threats targeting their application-layer interface. Since the HTTP protocol is a common denominator, powering numerous IoT devices, cloud-based services, application programming interfaces (APIs), and full-stack web applications in the IT, IoT, and even operational technology (OT) domains, it is an alluring target for malicious actors and security researchers alike. Such a volume and diversity of threats, combined with limitations in installing security mechanisms and proliferation of zero-day exploits, must be addressed with advanced security-based intelligence focusing on proactive detection of abnormalities [1,8], which can be established through well-thought-through combinations of technologies and methods, including the use of honeypots and honeynets, IDS/IPS systems, application-level gateways (ALGs), and web application firewalls (WAFs), and extended and enhanced with the power of machine learning (ML) and artificial intelligence (AI) algorithms. In this context, IoT and HTTP honeypots play a crucial role in capturing and analyzing incoming network requests, recording application-layer sessions and profiling visitors to distinguish legitimate users from botnet components, and serving crafted device or service models to deceive attackers while collecting data about their actions.

In order to capture and study the cyber threat landscape in the cloud, IoT, and web domain better, we propose, design, and implement a flexible HTTP honeypot system that can mimic any HTTP-based service or application. We position the developed honeypot platform as the main contribution of our paper and describe it in detail. We then validate its functionality by implementing 14 different honeypot services encompassing web applications (e.g., content management systems, database administration tools), Internet of Things devices (e.g., network cameras, network storage devices), and cloud services (e.g., database solutions, cloud infrastructure services), and running a 4-month-long experiment collecting attacker data. Next, we propose and demonstrate a novel honeypot-based mechanism for identifying internet scanning services and implement several data collection and enrichment approaches, including client-side browser fingerprinting, identification of Common Vulnerabilities and Exposures (CVEs), and request header profiling. Finally, we critically

evaluate the system's viability for profiling and classifying the attackers by demonstrating multiple use cases on the collected dataset.

1.1. State of the Art

We provide hereafter a short review of interesting implementations of low- and high-interaction honeypots and IDS and IPS systems, designed specifically for IoT, HTTP services, and cloud environments, with a focus on supporting different types of capabilities for cyberattack detection, monitoring, and collection of actionable cyber threat intelligence (CTI). However, it has to be noted that the dividing line between the IoT and cloud domains can be blurred, as both fields in many use cases converge on network protocols and infrastructure components and appliances, such as routers, switches, network attached storage (NAS), server-integrated Lights-Out (iLO) management interfaces, infrastructure controllers, uninterruptible power supplies (UPS), and the rest.

Low-interaction honeypots are typically designed for simple attacks and provide less actionable evidence for cybersecurity research [1]. Examples are numerous and extremely diverse, including e.g., Honeycam mimicking an Internet Protocol (IP) camera [1], IoT Honeypot exposing Telnet services [11], and HoneyIo-4 [12] capable of mimicking four types of IoT devices (a camera, a cash registry, a printer, and gaming console), and a honeynet built for a smart home IoT environment [13] mimicking a network video recorder, IP cameras, an IP door phone, and a set-top box in combination with a ViPNet IDS.

High-interaction honeypots mostly focus on mimicking entire devices rather than just selected protocols and services, and on the delivery of self-adaptive capabilities, allowing the honeypot to extend the range of behaviors adaptively and simulate different IoT devices to prolong the attack duration. Some examples of IoT honeypots capable of emulating entire devices and adapting their behavior are FIRMADYNE, ThingPot, IoTCandyJar, Chameleon, and Honware. FIRMADYNE is an extensible, self-adaptive, and automated framework capable of emulating commercial off-the-shelf (COTS) IoT devices based on an instrumented kernel. It features a Web crawler for firmware discovery, emulation capabilities, and dynamic analytics [14]. ThingPot was initially implemented to simulate an IoT platform with all supported application-level protocols, including, e.g., a Philips Hue smart lighting system [15], and was later extended with supervised machine learning (ML) for adaptive capabilities and distributed denial of service (DDoS) detection [16]. IoTCandyJar combines low and high interaction capabilities, exploiting ML with a Markov decision process for the discovery of the most suitable IoT device behaviors to extend attack sessions [17]. Chameleon is another self-adaptive IoT honeypot that utilizes a similar low- and high-interaction approach, which allows it to progressively improve responses for the devices that are being queried [18]. Honware is an adaptive high-interaction example, capable of simulating diverse IoT devices by processing a standard firmware image and extracting and adapting the file system accordingly [19].

Other examples of honeypot implementations include SIPHON [20], a scalable IoT honeypot platform designed specifically to mimic geographically dispersed IoT devices through wormholes, an ML-enhanced Cowrie implementation for Secure Shell (SSH) and Telnet observations [21], IRASSH-T, which is also focused on SSH/Telnet and presents adaptive capabilities based on reinforcement learning for reward function optimization [22], and a study that used three Cowrie honeypots to emulate an IoT system [23]. Numerous other examples can be found in the literature focusing on different types of new or extended honeypots to study various types of IoT cyberattacks, including Mirai botnet attacks, DDoS attacks, attacks only on SSH and/or Telnet, man-in-the-middle attacks, fileless malware attacks, and others [2]. Furthermore, numerous development efforts take the approach of training and extending the complexity of their frameworks by starting with a combination of a low-interaction honeypot and various types of scanning tools to discover, catalog, and fingerprint IoT devices and device families connected to the Internet, which is then followed by extending and fine-tuning the low-interaction honeypot into a high-interaction one. For example, a hybrid IoT honeypot framework, IoTCMAL [24], uses low-interaction

services such as SSH and Telnet in combination with high-interaction services hosted on physical devices and virtual private clouds. Researchers in [25] developed a honeypot framework U-PoT, focusing specifically on the emulation of UPnP-enabled IoT devices and extended with capabilities for automated IoT honeypot creation based on fingerprint documentation. Ref. [1] used IoTScanner for device discovery and IoTLearner to improve a low-interaction honeypot with an algorithm based on neural networks for more advanced response prediction. Other examples combine honeypots and honeynets with other tools, such as sandboxes and IDS systems. Ref. [26] proposes an IoT-BDA framework for automated IoT botnet detection, identification, analysis, and reporting by integrating honeypots with a sandbox, which allows for a wider range of supported configurations as well as additional capabilities to detect anti-analysis, persistence, and anti-forensics techniques. Similarly, IoTPOT [27,28] combines honeypot and sandbox capabilities to implement a medium-interaction trap exposing Telnet services and a malware sandbox emulating various embedded system environments. Honeynet is another well-known project involving high-interaction honeypots based on COTS IoT devices to complement IDS.

The design of a concrete IoT honeypot implementation is defined by the type of IoT devices we want to focus on and the range and type of cyberattacks that we want to address. Our research shows that the HTTP protocol and services are some of the most impactful attack vectors, for a number of reasons. Firstly, HTTP is one of the most widely used application-layer protocols on the Internet and adopted by many IoT devices for data transfer, cloud services, remote control, and user interaction through web interfaces, all of which are critical interactions that must be protected in order to prevent unauthorized access and maintain the confidentiality, integrity, and authenticity of the information being exchanged. Weak HTTP protection in IoT is often exploited, e.g., for establishing botnets, which facilitates malicious infection and control of IoT devices, malware spreading, and even organization of large-scale DDoS attacks, as well as for other cyberattack types. In fact, a web honeypot exposing HTTP endpoints can facilitate the detection of several different potentially malicious cybersecurity activities exploiting a broad range of tools and attack methods. However, in the realm of web applications, visitors may access a website for various intentions. An HTTP honeypot could receive organic visits from users mistyping the uniform resource locator (URL) of the service, receive periodic requests by scanners, crawlers, and bots performing site examination for indexing and data retrieval, or encounter malevolent users performing exploratory target analysis in an attempt to exploit the service. Therefore, research in this field makes use of user, attacker, and service modeling to design and deploy generic-service or IoT-specific HTTP honeypots, classify their visitors, and extract, monitor, detect, and study the activity of malicious actors. Moreover, HTTP honeypots bear the potential to distinguish advanced persistent threats from automated bot traffic by detecting targeted web scans and human actors performing heuristic service investigations during the reconnaissance phase preceding a potential attack [29]. Lastly, in addition to SSH/Telnet attacks, HTTP attacks are among the most common occurrences detected using network telescopes and IoT honeypots and honeynets [2,10], yet the reviewed literature shows that HTTP honeypots and honeynets continue to be underrepresented. Some examples of HTTP honeypot implementations are summarized in Table 1.

Table 1. Comparison of HTTP honeypot implementations.

Project	Honeypot Type	Level of Interaction	Supported Attack Types
Dionaea [5]	General	Medium	N/A
HoneyPy [30]	General	Low/Medium	N/A
IoTPOT [31]	IoT	Hybrid	Brute force attacks, Hajime, ZmEu attacks

Table 1. *Cont.*

Project	Honeypot Type	Level of Interaction	Supported Attack Types
Dowling [32]	IoT	Medium	Dictionary attack, brute-force attacks, reconnaissance attacks, botnet attacks, launch attacks, individual attacks
HoneyIo4 [12]	IoT	Low	Reconnaissance attacks
SIPHON [20]	IoT	High	Brute-force login attempts
Metongnon [10]	IoT	Low/Medium	Reconnaissance attacks
Scalable VPN-forwarded honeypots [33]	IoT	High	N/A
Lingenfelter [23]	IoT	Medium	Botnet attacks
Firmadyne [14]	IoT, entire device emulation		Reconnaissance attacks, buffer overflow
IoTCandyJar [17]	IoT, entire device emulation		HTTP (HEAD, OPTIONS, CONNECT), TCP, UDP, RTSP
Pandora [34]	General	Low	Login attempts, port scan attacks
Bartwal [35]	General	Hybrid	HTTP attacks (XSS, SQLi, OSC), DDoS, botnet attacks
X-POT [36]	IoT, entire device emulation	Hybrid	N/A
HoneyCamera [37]	IoT	Low	Login attempts, command injection, shellshock

1.2. Motivation

HTTP has, throughout the last two decades, emerged as a universal and widely used data exchange protocol. In addition to its applications in the IoT world, it also powers a broad range of cloud applications and application programming interfaces. Due to its popularity and the number of HTTP-based attacks and tools (including endpoint enumeration, URL and payload fuzzing, structured query language (SQL) and command injection, cross-site scripting, etc.), it has become a notable attack vector of many systems, be it consumer-facing, professional, or even mission critical. This is supported both in the scientific literature [38] and in our own research [39,40], where we observed large increases in unwanted and malicious traffic throughout the years, as well as a relative increase in HTTP-based traffic compared with other protocols. Thus, we reviewed and evaluated various HTTP honeypot implementations (Table 1) with the aim to deploy our own HTTP sensor network. However, since many of the aforementioned solutions were tailored to rather specific use cases, we decided on the development of our own honeypot system. We aimed to design a scalable and flexible HTTP honeypot platform that can mimic a wide variety of systems in addition to IoT devices. This includes HTTP-based cloud APIs as well as fully interactive web applications, designed to capture rich application-layer attack data that can be fused with data collected on the underlying network stack to provide comprehensive insight into attacker activity. For this, we aimed to bring together multiple technologies that are commonly used in other, non-cybersecurity-related domains, such as service modeling and data visualization, behavioral user analysis, advanced browser fingerprinting techniques, novel scanner service identification mechanisms, cross-layer data fusion, and open CTI data augmentation, enabling the creation of rich attacker profiles.

1.3. Structure of the Paper

The rest of the paper is structured as follows. Section 2 describes the proposed system in terms of identified requirements, architecture, and actual implementation. It also outlines our proposed approach towards the selection of relevant services to be modeled as honeypots and describes some of the already supported technologies. Section 3 describes

experimental results, including the experiment setup, structure of the data set, and different applications that validate the approach. These include crawler and bot detection, together with a novel approach towards crawler identification, browser and operating system fingerprinting, identification of the used Common Vulnerabilities and Exposures (CVEs), and different data visualizations. In Section 4, we briefly discuss some of the encountered challenges, design decisions made, and lessons learned during the platform development and deployment. Lastly, in Section 5, we critically assess the advantages and disadvantages of the presented approach, conclude our work and outline prospects for future work.

2. Proposed System

In this section we outline the system requirements, and, based on these, design the system architecture and implement a scalable and flexible platform for capturing and analyzing attacker traffic. The proposed design has been iteratively refined throughout multiple research and development projects and is based on a number of requirements, engineering sessions, and discussions with national cybersecurity-related stakeholders.

2.1. Requirements Identification

Based on the literature review [41–43], we identified several stakeholder groups that would benefit from such a solution in the space of Internet of Things, and cloud and web services, and held multiple discussions and interviews at the national level. Additionally, we performed an extensive review of the literature and funded research projects to identify further requirements, which led to a high-level list of 35 functional and 7 non-functional requirements. Functional requirements are further categorized into 9 subgroups (communication, protocols, models, logging, data enrichment, data analysis, dynamic response, user interface, and data export), while non-functional requirements encompass the architectural decisions and deployment strategy, as well as the detection, usability, and security of the solution. Requirements were then prioritized as: (1) high priority—necessary implementation for basic use cases; (2) medium priority; and (3) low priority—possible later implementation for specialized use cases. All requirements are shown in Figure 1.

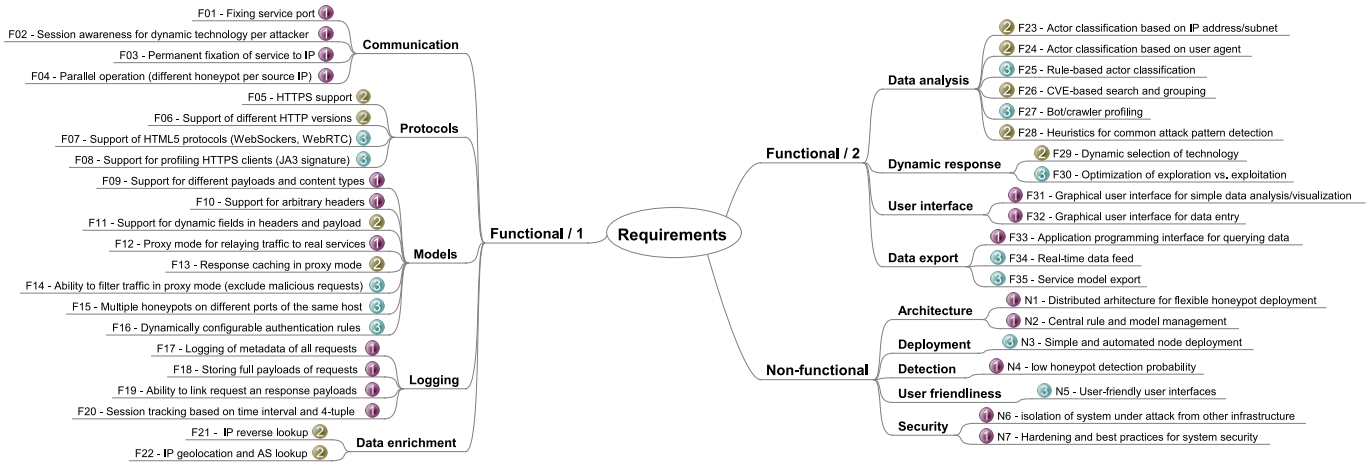


Figure 1. Identified functional and non-functional requirements. Functional requirements are categorized into 9 groups: communication, protocols, models, logging, data enrichment, data analysis, dynamic response, user interface, and data export. Non-functional requirements encompass architectural decisions, deployment strategy, detection, usability, and security of the solution. Priority is denoted with a number next to each requirement. Priority 1 is the highest and priority 3 is the lowest.

2.2. System Architecture

We approached the platform design in accordance with the identified requirements. We prioritized three crucial aspects narrating the system architecture: high scalability in

terms of deployed honeypots and supported services (allowing us to deploy new sensors swiftly in diverse environments), adaptability (enabling effortless refinements of honeypot responses and allowing quick adaptation to the changing cybersecurity landscape), and efficient data collection to facilitate analytics and enable fast learning. The integrated scalability thus enhances the system's coverage, and a dynamic and adaptable honeypot response mechanism ensures the system's agility and augments its responsiveness to emerging threats, while the centralized data collection pipeline enables data aggregation and analysis in a unified manner.

Accordingly, we implemented a robust microservice-based platform, combining distributed honeypot units with centralized logic, management, and data collection functionalities. For efficient communication between the honeypot units, the central control, and the central data store, we favored robust communication mechanisms. Remote honeypot units communicate with the central control system using representational state transfer (REST) APIs with added local caching to ensure performant responses and maintain a sufficient degree of autonomy. We employed log shipping to ensure the delivery of new data to the central data processing system and facilitate streamlined data processing. In summary, the devised architecture design consists of three integral components (see Figure 2):

- Honeypot units, which receive requests from attackers and form replies based on the provisioned behavior; these can operate different modes, either autonomously, or by proxying the attackers' requests to the central control system, where responses can be prepared with greater control.
- A central control system with APIs, databases and control logic; it registers and manages honeypot nodes, acts as the source of the latest response models, and in the proxy mode continuously receives requests from the honeypot units and prepares responses according to the content of the request.
- The data pipeline, which ships all request logs to a storage cluster, distributes the metadata to enrichment workers, and runs analyses.

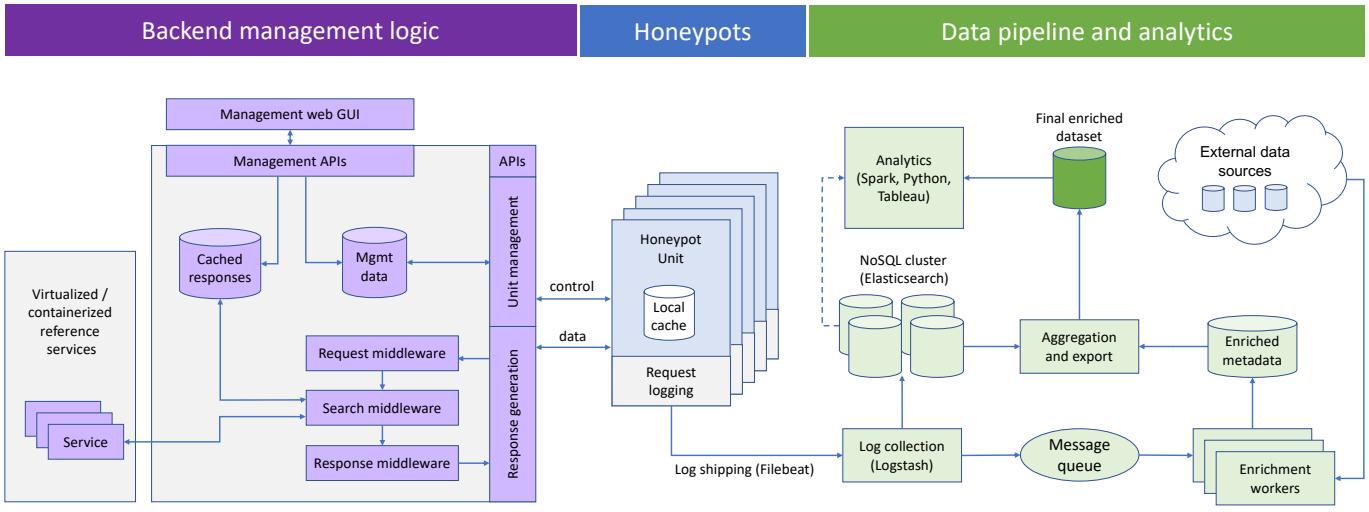


Figure 2. High-level architecture of the distributed honeypot system.

2.2.1. Central Management

The central management server is responsible for remote honeypot configuration, as well as for generating responses to individual honeypot nodes. As such, it represents a reference point for all models, and it allows the data to be centrally upgraded. Responses to requests are propagated to final honeypot nodes, where they are cached; this can greatly improve performance in the case of distant nodes (e.g., a different continent, where each request would require a long round-trip time). In the first approximation, we assume that the models will not change so quickly that a different way of distributing them to the honeypot nodes would be necessary.

The central back-end can generate a response in two ways, depending on the chosen technology. The first, low-interaction, mode of operation relies on answers for the selected technology that are statically stored in the database. In this case, the answers must be previously enumerated and added to the collection, which is possible either manually, or in an automated way by crawling a service. In both cases, we want the ability subsequently to modify certain fields dynamically in the response (e.g., current date, time, random strings, blockchain addresses, etc.), depending on the technology configuration. The second mode of operation can yield high-interaction honeypots and relies on dynamically generated responses based on proxying a request to a real reference service. This mode can support arbitrarily complex services but requires access to the actual software behind the service.

2.2.2. Honeypot Nodes

The honeypot node acts primarily as a specialized and highly configurable HTTP proxy, with the configuration being remotely managed by the central system; however, if need be, the configuration can also be stored locally to make the honeypot autonomous. The honeypot node HTTP server also performs the TLS termination, but does not answer any requests directly; all of the HTTP requests are proxied, either to a central system, which can serve static (cached) responses from a database of already modeled services, or to a virtualized/containerized reference service, which can be either centrally deployed or co-located on the node host. On the data collection part, the honeypot node HTTP server logs all incoming requests to a rotating log file that is shipped to central storage.

2.3. Threat Impact Assessment for Service Identification

The usefulness of the platform is directly dependent on the types of supported honeypots. The potential impact of any findings based on the collected data for a particular modeled service increases with the service's popularity, the number of its known vulnerabilities, and the estimated cost of the breach, but can be reduced by appropriate countermeasures and security compliance.

Initially, we chose a number of technologies based solely on popularity, such as WordPress, Joomla, PhpMyAdmin (web apps), several IoT devices (HikVision webcam, HP and Epson networked printers), cloud infrastructure (VMWare vSphere Server, OpenStack, MongoDB), storage appliances (various network-attached storage systems), and others. However, to make the selection more systematic, we designed a metric called the cyber-threat impact score [44], which could be used to prioritize the supported honeypots even better:

$$cyberthreat\ impact\ score = \frac{V_d \times B_c}{C_e \times C_i} \times U_c, \quad (1)$$

Here, V_d represents the vulnerability density of the app, B_c is the estimated breach cost, C_e is the effectiveness of the countermeasures, C_i represents the compliance index, and U_c is the real-world install base. These factors can be difficult to determine in practice and are subject to actual use case, intent, and capability of the owner, and number and availability of deployed services or devices. Nonetheless, by making certain assumptions and simplifications to the formula, a usable score to rank the technologies can be obtained:

- To estimate the vulnerability density, V_d, we first need to estimate the code base size, which is easily doable for open-source projects only; for closed-source projects, the code base size has to be approximated with a similar project in terms of functionality;
- For the number of known vulnerabilities, the first approximation can be obtained by querying a CVE database;
- To estimate the breach cost, we can first estimate the risk level of the application (low-risk applications with insignificant breach consequences to mission-critical apps with severe service disruption as a consequence) and map it to the interval from 0 to 1; for this, we propose to use the sigmoid function; however, the initial risk should be estimated on a per-use-case basis, which makes this impractical. To simplify the

calculation, the factor can be assumed as a constant of 0.5 and, if need be, adjusted upwards for intrinsically risky applications, or downwards for less risky applications.
- Effectiveness of countermeasures could be estimated based on lookups in countermeasure databases such as [45]. This factor then signifies the average countermeasure effectiveness for known vulnerabilities.
- The existence of countermeasures represents no guarantee that these countermeasures are implemented. To get around this, we can assume the compliance index factor is a constant, signifying that all known countermeasures are implemented, or that none are.
- Finally, the installed base of the application could be estimated using market research, download counters, etc. However, the most reliable and consistent data in our experience come from internet surveys, obtained from scanners such as Shodan, Censys, and similar.

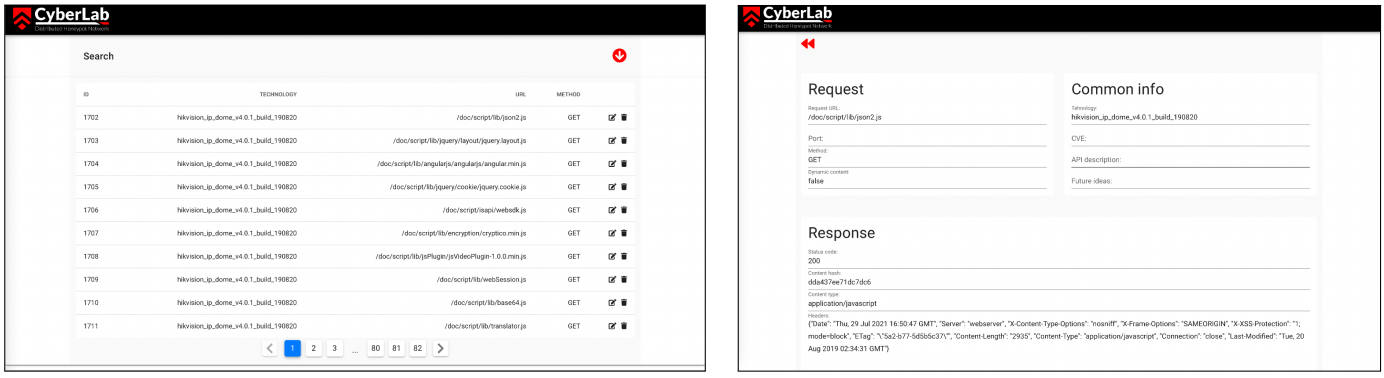
2.4. Implementation Details

In the following subsections, we provide the details regarding the practical implementation of the system. We discuss our main considerations and disclose the rationale behind our choices.

2.4.1. Central Management

The central control system stores the canonical state of all configured services and generates responses to the underlying honeypot units based on the configurations. The system is implemented in Python and includes a relational database with honeypot configuration data, along with basic request statistics that are needed for decision logic. The honeypot configuration data include the emulated device or service technology, service port, hit counter, and a link to the service model table. The latter stores the honeypot URL endpoints with the corresponding service responses, their counter, and headers. In addition, a fast in-memory Redis store with data persistence is used to store all service assets (HyperText Markup Language (HTML) bodies, Cascading Style Sheets (CSS) and JavaScript (JS) files, and binary files) required for the honeypot operation.

The central system exposes two sets of APIs, one for the honeypot operation and another for the management interface, where settings and payloads can be manually edited (Figure 3). This is useful for streamlined manual inspection of responses, enabling on-the-fly adaptability. The interface allows for the replacement of static text with dynamic fields (e.g., date and time), as well as other modifications, such as the inclusion of client-side profiling code (e.g., cookies, local storage tokens, and advanced JavaScript-based client fingerprinting mechanisms).



(**a**) (**b**)

Figure 3. Response management graphical user interface. (**a**) Listing of endpoints for a specific service. (**b**) Editing of a specific response with HTTP headers and body.

2.4.2. Honeypot Nodes

Distributed honeypot nodes listen for incoming network connections, forward received HTTP requests to the central control system, cache its responses, and serve them to end users. Simultaneously, they locally log all communications and ship the logs to the centralized data store.

The honeypot units were developed in Python and rely on the FastAPI framework and Uvicorn server, among other dependencies. They consist of 5 core modules handling unit configuration, technology (service) selection, attacker requests, request logging and local log rotation, and response caching. They are deployed as a composition of Docker containers including the honeypot server unit, a local Redis store for caching, and a FileBeat container for log shipping. The units support HTTP and HTTPS operation and are capable of listening concurrently on multiple Transmission Control Protocol (TCP) ports. This is achieved using Docker's port-forwarding options, mapping multiple exposed host ports to a single port on the container. This effectively mirrors the traffic and reduces central processing unit (CPU) and memory usage by sparing the listener process of creating a new server instance for each of the ports. Nevertheless, the honeypot units still preserve information about the originally targeted ports, and additionally also generate a communication session identifier, allowing for subsequent analysis of all the associated traffic within a given time window.

Lastly, we developed a novel approach toward web scanner service identification and embedded its code into the honeypot units. Following our approach, each unit essentially "watermarks" all outgoing responses by steganographically concealing the information on the requesting client (e.g., their IP and user agent) in the response headers and payload itself. This enables us to identify the source of the traffic when a particular response is found in any of the (public) scanner databases. We further describe this mechanism in Section 3.3.

2.4.3. Data Pipeline

All raw data are collected centrally in an ElasticSearch cluster. The data delivery process is accomplished using local Filebeat agents on the honeypot units, which forward the data to the Logstash-based filtering and distribution component. The latter not only stores the data in the Elasticsearch cluster but also forwards them to the RabbitMQ message queue, serving as a near-real-time delivery mechanism for all subscribers. The subscribers in the system are the data enrichment, enumeration, and summarisation utilities that query the external CTI databases for each newly encountered value, and keep track of data counters and online statistics.

Data enrichment, linking, and filtering components store their results in a relational database; this includes resolving the source IP addresses to geolocation objects, resolving the autonomous system (AS) information, querying IP and Domain Name System (DNS) reputation databases (e.g., blocklists, abuse reports), cross-matching entries in structured CVE databases, and classifying the traffic sources (VPN, Tor, proxy lists).

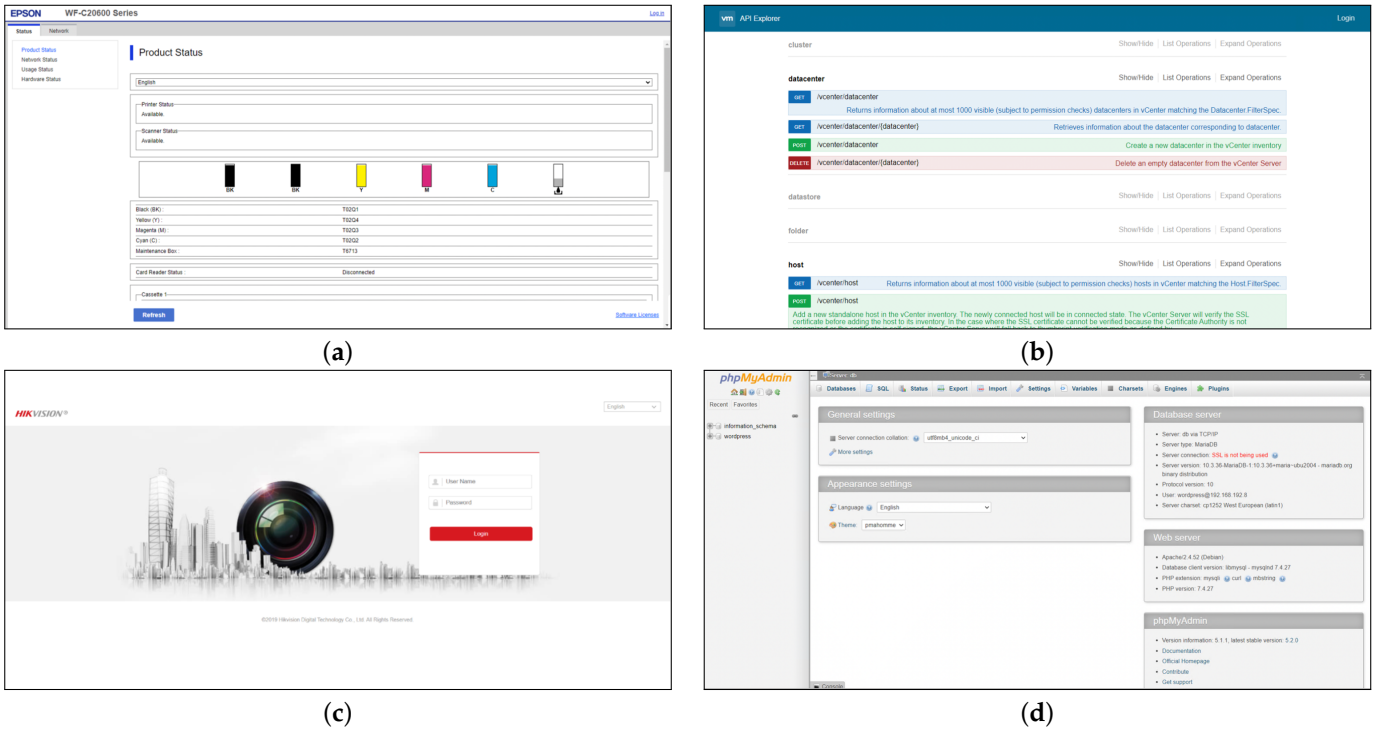
Finally, relevant data from all databases are periodically aggregated and exported into a portable SQLite format. We find this the most flexible format that is well-supported in various analytics tools. Nonetheless, certain data analysis tools may also directly query and extract source data from the centralized Elasticsearch cluster and the accompanying SQL and NoSQL support databases containing additional resources and aggregate data.

2.4.4. Implemented Honeypots

Device or service honeypot implementation strategies depend on their complexity. Simple low-interaction web services are crawled in an automated manner and their responses are stored in a document database. Each service is then inspected for the presence of potential dynamic fields (e.g., IP addresses, various identifiers, time and date, blockchain addresses), which are replaced with dynamic variables updated by the response middleware during every request. Such low-interaction service models can be iteratively refined by listing requests without a matching response rule or by searching for anomalies in request referer trees, re-crawling, and adding new endpoints to the dataset. Using both of

these techniques, a large number of simple services (e.g., APIs, simple IoT devices, printers, modems, simple web interfaces, login screens, and status pages, etc.) can be modeled.

High-interaction honeypots, on the other hand, are implemented as transparent pass-through proxies intercepting and forwarding traffic to real containerized or virtualized reference services or IoT devices in secure environments, for the sole purpose of providing accurate dynamic responses whilst capturing the session data. Complex services, devices, and services under detailed observation can be deployed in this manner to preserve their internal states and thus act more convincingly. This method is mostly suitable for deploying honeypots of complex open-source software (e.g., phpMyAdmin, WordPress, etc.), but can in practice be used with any software or product (Figure 4).



(**a**) (**b**)

(**c**) (**d**)

Figure 4. Examples of implemented honeypots. (**a**) A full scrape of an EPSON printer with added dynamic fields. (**b**) A full scrape of a VMware vCenter Server management GUI. (**c**) A full scrape of the HikVision web camera with added dynamic fields. (**d**) Proxy mode honeypot to a real containerized phpMyAdmin instance.

3. Experimental Validation and Results

3.1. Experiment Setup

The experiment was set up to run from 3 March to 14 July 2023 (a total of 132 days). During that time, a total of 14 honeypot services were deployed on 14 separate IP addresses in multiple networks. The IP addresses were not actively advertised in any database, forum, or web page. The deployed honeypots comprised three different types: Internet of Things, cloud services, and classic web applications. They offered various degrees of interactivity. Low-interaction honeypots were used to emulate simple devices and web applications consisting mostly of static websites. Where necessary, these were additionally augmented with dynamic variable data to provide a higher level of interaction and convincingly emulate real services. High-interaction honeypots transparently proxied all communication to containerized complex web applications, thus preserving their state. The deployed honeypots, along with their type and the level of interaction they provide, are listed in Table 2.

Table 2. Deployed honeypots, their type, and their provided level of interaction. Low-interaction honeypots augmented with dynamic data variables are categorized by their provided level of interaction as Low/Medium.

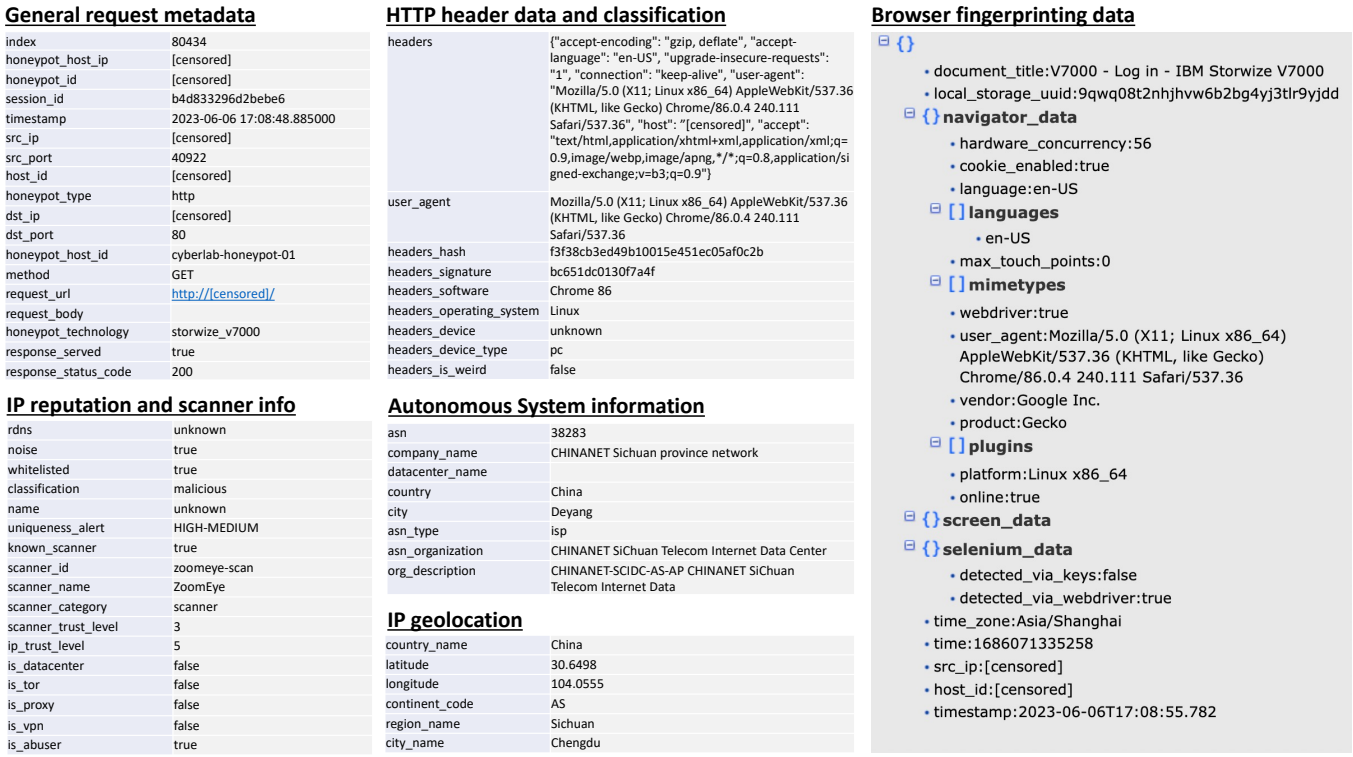
Honeypot	Honeypot Type	Level of Interaction
EPSON c20600, printer	IoT	Low/Medium
HP Color LaserJet m552, printer	IoT	Low/Medium
APC SmartUPS, an uninterruptible power supply	IoT, Cloud	Low/Medium
MongoDB v2.4	Cloud	High
MongoDB v3.2	Cloud	High
Openstack v17	Cloud	Low/Medium
VMware vCenter Server v6.5	Cloud	Low/Medium
QNAP Network attached storage system	IoT, Cloud	Low/Medium
QSAN Network attached storage system	IoT, Cloud	Low/Medium
Thecus Network attached storage system	IoT, Cloud	Low/Medium
IBM Storwize v7000, enterprise storage system	IoT, Cloud	Low/Medium
phpMyAdmin v5.1.1	Cloud, Web	High
Joomla CMS v3	Web	High
Joomla CMS v4	Web	High

3.2. Acquired Data

The collected datasets include metadata captured at the network, transport, and application layers, including all headers of IP packets, TCP segments, and HTTP requests and responses. To capture the data on IP and TCP layers, p0f was used in parallel with the honeypot unit's web server. Such cross-layer visibility allowed us to obtain the TCP segment and IP packet information, namely the TCP window size and IP Time-to-Live, which can give hints about the used operating system versions. The data collected at the HTTP layer included all headers, URLs, methods used, and the request body. For each request, we also logged whether the response was known or missing, so iterative honeypot improvements were possible. All collected data were timestamped for calculation of the time to first contact and the relapse time to recurrent attacks. The general collected request metadata are displayed in the first section of Figure 5.

All captured and enriched request metadata are listed in subsections of Figure 5. The data were enriched in a number of ways. Firstly the IP reputation was obtained from a number of publicly available databases and its reverse DNS record was correlated with DNS blacklists. The IP reputation metric included various classifications, such as whether the IP belongs to a data center, is used as a Tor exit node, is a known proxy, or has been previously reported for abuse. In addition to that, Internet scanning services were identified using multiple methods, as described in more detail in Section 3.3. Next, the autonomous system information was obtained from publicly available databases, yielding the owner of the IP address space. The IP address was also geolocated to obtain approximate visitor location, including country, city, and longitude and latitude, where available. We also enriched the request header information using open databases, such as a database of known user agent (UA) strings to determine whether the UA came from a legitimate web browser, a network utility, a software library, or whether it was generated by a poorly written algorithm in an attempt to mimic a legitimate client. Trivial detection of the client's operating system and software version was also performed by analyzing the UA string. Finally, we also developed a powerful, client-based browser fingerprinting system to obtain all available client capabilities, record and forward information on user-generated events, and place persistent visitor identification tokens in the client's local storage. This was possible only

if the client-side JavaScript was interpreted, which excluded crude web scraping and reconnaissance tools, but yielded useful results for the detection of browser automation scripts and human actors, potentially indicating advanced persistent threats. We describe this system in more detail in Section 3.4.



General request metadata

index	80434
honeypot_host_ip	[censored]
honeypot_id	[censored]
session_id	b4d833296d2bebe6
timestamp	2023-06-06 17:08:48.885000
src_ip	[censored]
src_port	40922
host_id	[censored]
honeypot_type	http
dst_ip	[censored]
dst_port	80
honeypot_host_id	cyberlab-honeypot-01
method	GET
request_url	http://[censored]/
request_body	
honeypot_technology	storwize_v7000
response_served	true
response_status_code	200

HTTP header data and classification

headers	{"accept-encoding": "gzip, deflate", "accept-language": "en-US", "upgrade-insecure-requests": "1", "connection": "keep-alive", "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4 240.111 Safari/537.36", "host": "[censored]", "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"}
user_agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4 240.111 Safari/537.36
headers_hash	f3f38cb3ed49b10015e451ec05af0c2b
headers_signature	bc651dc0130f7a4f
headers_software	Chrome 86
headers_operating_system	Linux
headers_device	unknown
headers_device_type	pc
headers_is_weird	false

Browser fingerprinting data

- {}
  - document_title:V7000 - Log in - IBM Storwize V7000
  - local_storage_uuid:9qwq08t2nhjhvw6b2bg4yj3tlr9yjdd
  - {} navigator_data
    - hardware_concurrency:56
    - cookie_enabled:true
    - language:en-US
    - [] languages
      - en-US
    - max_touch_points:0
    - [] mimetypes
    - webdriver:true
    - user_agent:Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4 240.111 Safari/537.36
    - vendor:Google Inc.
    - product:Gecko
    - [] plugins
    - platform:Linux x86_64
    - online:true
  - {} screen_data
  - {} selenium_data
    - detected_via_keys:false
    - detected_via_webdriver:true
  - time_zone:Asia/Shanghai
  - time:1686071335258
  - src_ip:[censored]
  - host_id:[censored]
  - timestamp:2023-06-06T17:08:55.782

IP reputation and scanner info

rdns	unknown
noise	true
whitelisted	true
classification	malicious
name	unknown
uniqueness_alert	HIGH-MEDIUM
known_scanner	true
scanner_id	zoomeye-scan
scanner_name	ZoomEye
scanner_category	scanner
scanner_trust_level	3
ip_trust_level	5
is_datacenter	false
is_tor	false
is_proxy	false
is_vpn	false
is_abuser	true

Autonomous System information

asn	38283
company_name	CHINANET Sichuan province network
datacenter_name	
country	China
city	Deyang
asn_type	isp
asn_organization	CHINANET SiChuan Telecom Internet Data Center
org_description	CHINANET-SCIDC-AS-AP CHINANET SiChuan Telecom Internet Data

IP geolocation

country_name	China
latitude	30.6498
longitude	104.0555
continent_code	AS
region_name	Sichuan
city_name	Chengdu

Figure 5. Collected and enriched request and IP metadata constituting a rudimentary attacker profile based on their IP address, header information, and client-side browser fingerprinting. Sensitive information was redacted from the figure.

During the course of the honeynet operation, we captured more than 188,000 HTTP requests. We identified 107,954 unique visitor sessions, corresponding to 17,176 unique IP addresses originating from 26 different autonomous systems. We identified that nearly 25% of all requests originated from 3160 different web crawler IP addresses, corresponding to 26 search engines. Nearly 30% of all source IP addresses originated or relayed malicious traffic. Less than 1% of visiting IP addresses interpreted JavaScript code. Out of those, we identified at least 21 bots that used known browser automation software and 15 visitor IP addresses with client-side event patterns, indicating a high probability of human actors. Key figures about the captured dataset are listed in Table 3.

Table 3. Key figures about the captured dataset.

Length of Data Collection Interval	132 Days (3 March 2023 to 14 July 2023)
Number of events collected (HTTP requests)	188,287
Number of unique sessions	107,954
Number of unique source IP addresses	17,176
Number of unique source ASNs	1807
Total number of identified search engines in the dataset	26

Table 3. *Cont.*

Total number of IP addresses of identified search engines	3160 (18.4% of all captured IP addresses)
Total number of requests from identified search engines	45,353 (24.1% of all requests)
Total number of ASNs of identified search engines	39 (2.2% of all captured ASNs)
Total number of deployed honeypots	14
Number of unique user agent strings	3227
Total number of IP addresses with malicious traffic	5006 (29.2% of all captured IP addresses)
Total number of source ASNs with malicious traffic	1327 (73.4% of all captured ASNs)
Total number of returning visitors [1]	7687
Total number of unique IP addresses that loaded JavaScript	159 (0.9% of all unique IP addresses)
Total number of sessions that loaded JavaScript	232 (0.2% of all captured sessions)
Total number of returning visitors that loaded JavaScript	40
The largest number of returning visits by an IP that loaded JS	14 (6.0% of all JS-enabled sessions)
Total number of detected IP addresses using browser automation tools (based on client-side fingerprinting)	21 (13.2% of all JS-enabled visitor IPs)
Total number of IP addresses with a high probability of human actors (based on client-side fingerprinting)	15
Total number of returning IP addresses with a high probability of human actors (based on client-side fingerprinting)	2 (2 sessions by one IP, 3 by the other)
Total number of captured client-side events	26,502

[1] A visitor is counted as returning if there are at least two registered sessions with the same source and destination IP; a session expires after 10 min of inactivity.

3.3. Crawler and Bot Identification

A significant amount of captured traffic comes from bots and crawlers, either from white-hat or grey/black-hat actors. We wanted to identify such requests to separate them from the actual attacks and further determine which of them only performed benign activities, as opposed to potentially harmful/malicious actions.

Through a review of the literature and related research, as well as lookups based on the collected IP addresses, we assembled a list of 48 known scanners. Throughout the entire data collection process, we were able to map 9955 IP addresses to these services. To do that, we formed a heuristic method based on four techniques:

1. We manually identified well-known security scanners, search engines, and bots, and, where provided, used the source IPs and subnets to label the requests (e.g., GoogleBot);
2. We performed a reverse DNS lookup for each IP, labeling the whitelisted scanners based on the resolved domain;
3. We used specially crafted replies that steganographically stored the traffic source information in the HTTP response, which effectively watermarked the responses. Searching for our honeypot IPs on well-known vulnerability scanner databases then often yielded enough of the actual watermark such that we were able to recover the source of the scans;
4. We amended the data using external services and databases.

The identified IP addresses were cumulatively responsible for 24.1% of all requests, which yielded a significant improvement in analysis. Additionally, cross-checking the IP information obtained with the above heuristics revealed that many attackers spoof the user agents of well-known services, such as the Baiduspider or GoogleBot. However, we also detected cases of malicious requests originating from legitimate scanning services, namely from ZoomEye, Fofa, and LeakIX. Moreover, even some of the IP addresses associated with legitimate services, as well as some web providers, were caught originating or propagating

malicious requests. These examples included the Google Global Cache infrastructure, co-located in regional internet service provider networks, as well as various web link preview services in chat clients, such as Skype and Telegram, regularly fetching previews of unfiltered user-supplied links from their datacenter IP addresses, thus proxying the potential URL parameter injection attacks.

In Figure 6 we break down the detected scanning services according to categories encompassing desktop browsers of several operating systems, popular scanning tools, and programming languages. Their categories were determined using the self-reported user agent field, meaning they are subject to spoofing and thus have to be verified against other features and metrics.

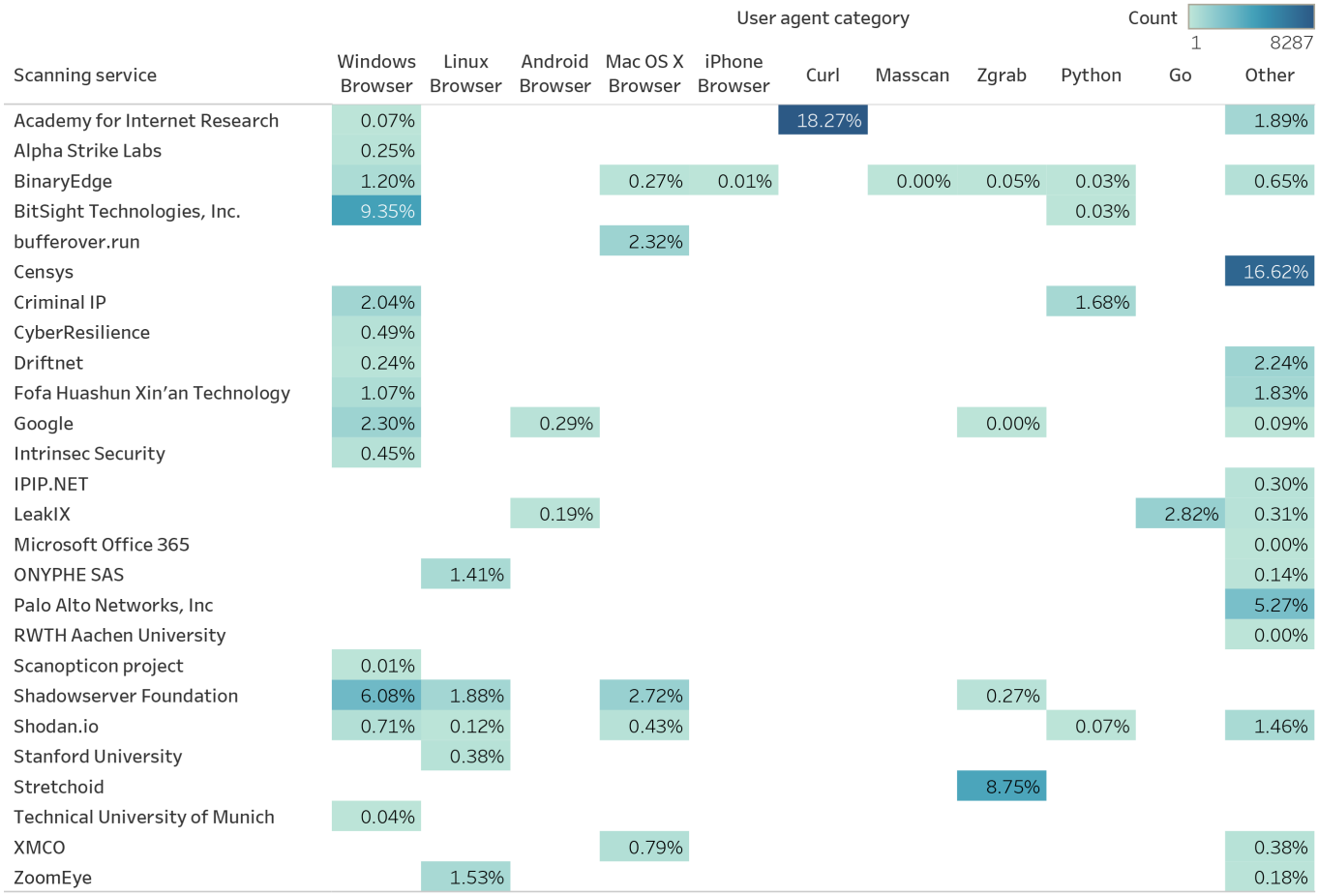


Figure 6. Scanning services categorized into groups of operating systems browsers, scanning tools, and popular programming languages, based on the self-reported user-agent HTTP header field. The following categories are missing from this visualization, as they were not used by any scanning services in our observed 132-day interval: BlackBerry Browser, FreeBSD Browser, iPad Browser, Java, Log4Shell Exploit, NetBSD Browser, Nokia Phone, OpenBSD Browser, OS/2 Browser, Perl, Roku, SonyEricsson Phone, and SunOS and WebOS Browser.

3.4. Client-Side Browser Fingerprinting

Sophisticated attackers may employ techniques for the artificial assembly of HTTP requests to imitate legitimate user activity and simulate browser behavior that is sufficient to bypass modern security checkpoints, such as web application firewalls; however, this has become increasingly difficult. Modern cloud application firewalls and bot protection mechanisms today often complement advanced server-side client analysis, such as Transport Layer Security (TLS) fingerprinting, with advanced client-side browser fingerprinting, which can become nearly impossible for attackers to emulate due to the complexity of mod-

ern browsers. While JavaScript execution may be a quick litmus test for the detection of real users, modern browser APIs expose the internal state of the client and can reveal further details on the device and its user. This includes details on the device's screen resolution, information about the device's graphics processing unit and canvas rendering characteristics, the number of its CPU cores, device sensors, and sensor inputs, connected external media devices, operating system version, user locale and timezone, installed system fonts, installed browser plugins, details on and access to in-browser storage, and precise information on user events, such as mouse movements, key presses, clicks, tab changes, etc. Consequently, attackers often resort to full browser emulation and use browser automation tools in order to bypass these fingerprint-based bot detection mechanisms. To facilitate the detection of such automation tools, we complemented our honeypot-collected metadata with a developed client-side fingerprinting process to determine whether browser automation was used and profile user actions to estimate the probability of human actors.

Our solution is deployed as an obfuscated JavaScript code, served by the honeypots and concealed within their every response. The script captures more than 66 client parameters, installs a persistent visitor identifier, and listens for over 16 browser events, including all user actions. The captured data are timestamped and periodically uploaded back to the original honeypot server in a concealed manner, simulating the refresh of a dynamic element on the website. The solution, combined with honeypot-retrieved visitor metadata, enables advanced user profiling and allows for the detection of web browser automation software, such as Selenium [46], Puppeteer [47], Playwright [48], and others.

Using heuristic analysis of the collected data, we attempted to model legitimate user behavior by exploring the distribution, timing, and correlation of the triggered client-side events. A sample session visualization of what we believe to be a human attacker on one of the deployed honeypots is depicted in Figure 7. The left side of the figure portrays the distribution of the recorded session events, while the right side depicts the event timeline relative to the initial site visit. As noticeable, pointer movements constitute the majority of the recorded user events, followed by scrolls and clicks. Browser-fired events, such as blur and focus, correlate with (in)activity periods, indicating that the user switched focus between open windows during the session. Lastly, the session duration, albeit only spanning 120 s, remains vastly longer than the majority of bot-orchestrated visits.

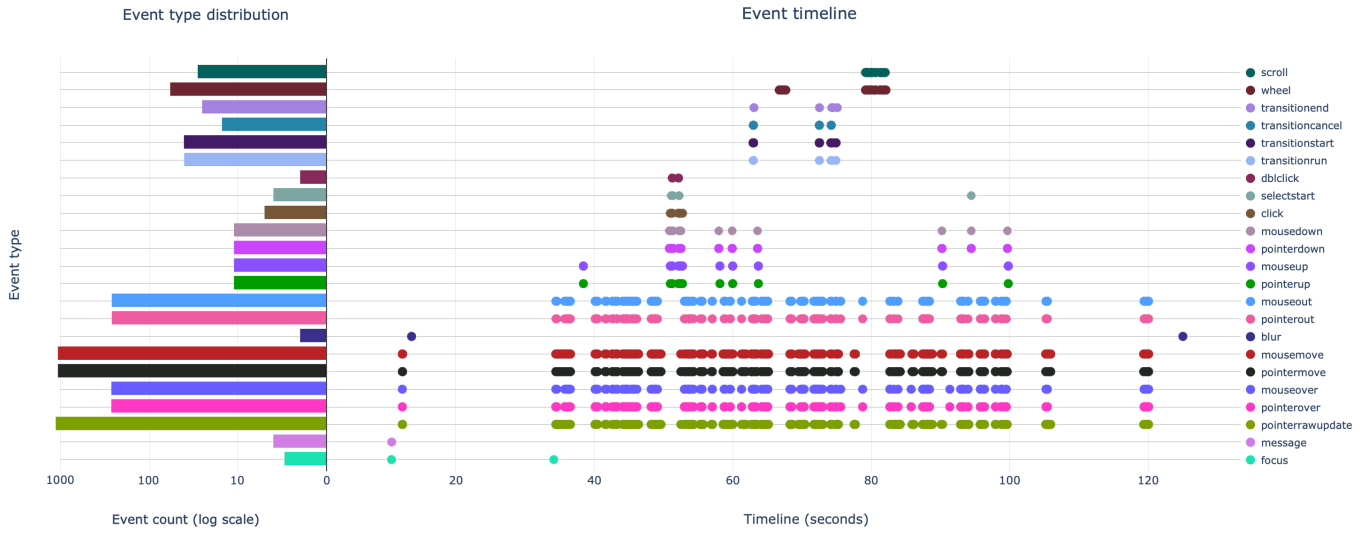


Figure 7. Visualization of the user session on a deployed honeypot based on the captured client-side events. The left side of the figure portrays the distribution of the recorded session events, while the right side depicts the event timeline relative to the initial site visit. Recorded event types are listed in a legend on the far right side of the figure.

Following the heuristic analysis of randomly sampled honeypot sessions, we constructed a rudimentary single-dimensional representation of client event distribution (discarding timing data and inter-event correlations) to demonstrate a simplistic approach toward user/bot modeling based on the collected data. The captured event distribution is depicted in Figure 8. Gray columns depict the occurrence of every captured client event across all of the recorded sessions. Red columns depict client event distribution for 21 bot sessions involving known browser automation tools, which we detected using specific client parameters. Blue columns depict event distribution for 19 sessions of what we believe to be real (human) users.

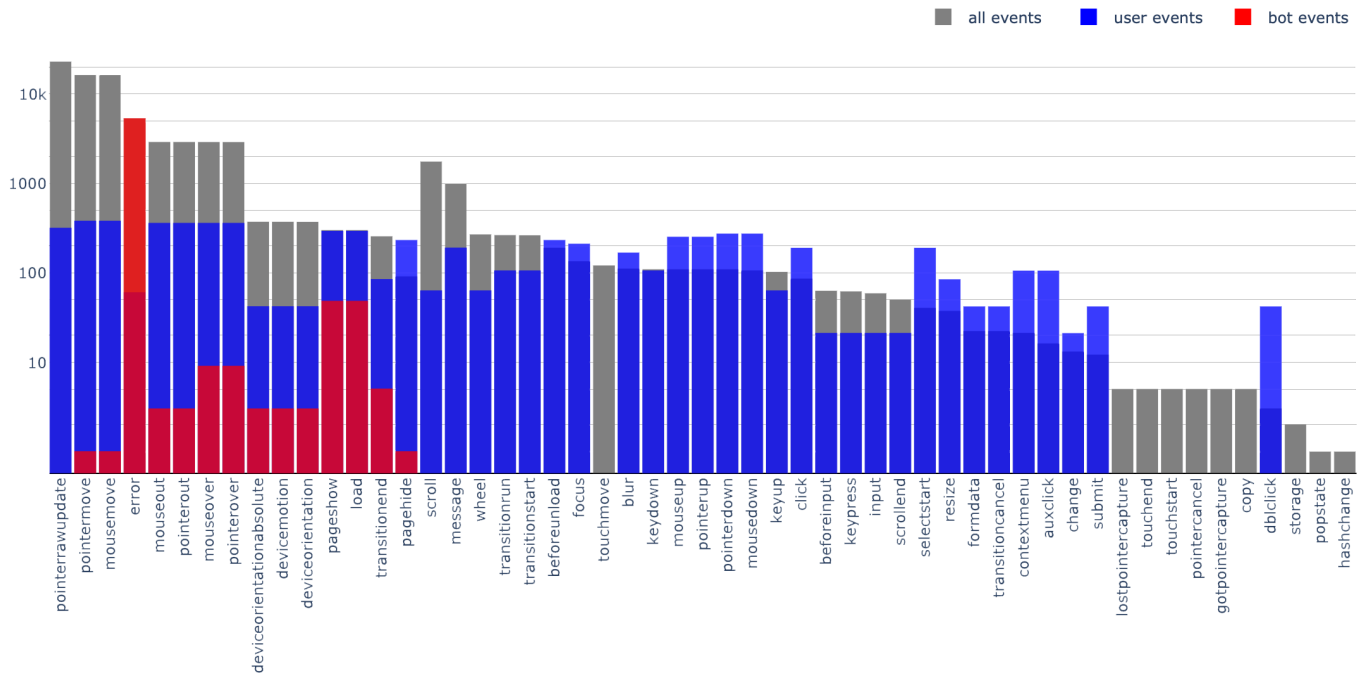


Figure 8. Visualization of client-side event distribution for all 232 captured sessions (gray), 21 sessions of known bots using browser automation software (red), and 19 sessions of what we believe to be real (human) visitors, as a rudimentary approach toward behavioral user/bot modeling. Event types are listed on the X-axis, while their counts are shown on the Y-axis using a logarithmic scale.

As evident from Figure 8, the detected bots only triggered a limited and volumetrically smaller subset of browser events otherwise encountered during the rest of the recorded sessions. However, interestingly enough, a number of automated browsers triggered device orientation changes and device motion events. Likewise, not all events were triggered, even by suspected human visitors, indicating that none of the select visitors used a mobile device to access the honeypots. However, we did manually investigate the suspected user sessions and supported them with additional metrics, including measuring session time and visualizing the locations of clicks to validate their interaction with honeypots. Lastly, we constructed graph visualizations of event transitions for all captured sessions. Upon their review, we identified that the bots used extremely modest event sequences compared with their human counterparts. Visualization of event transition chains for multiple sessions of a particular bot is depicted in Figure 9.

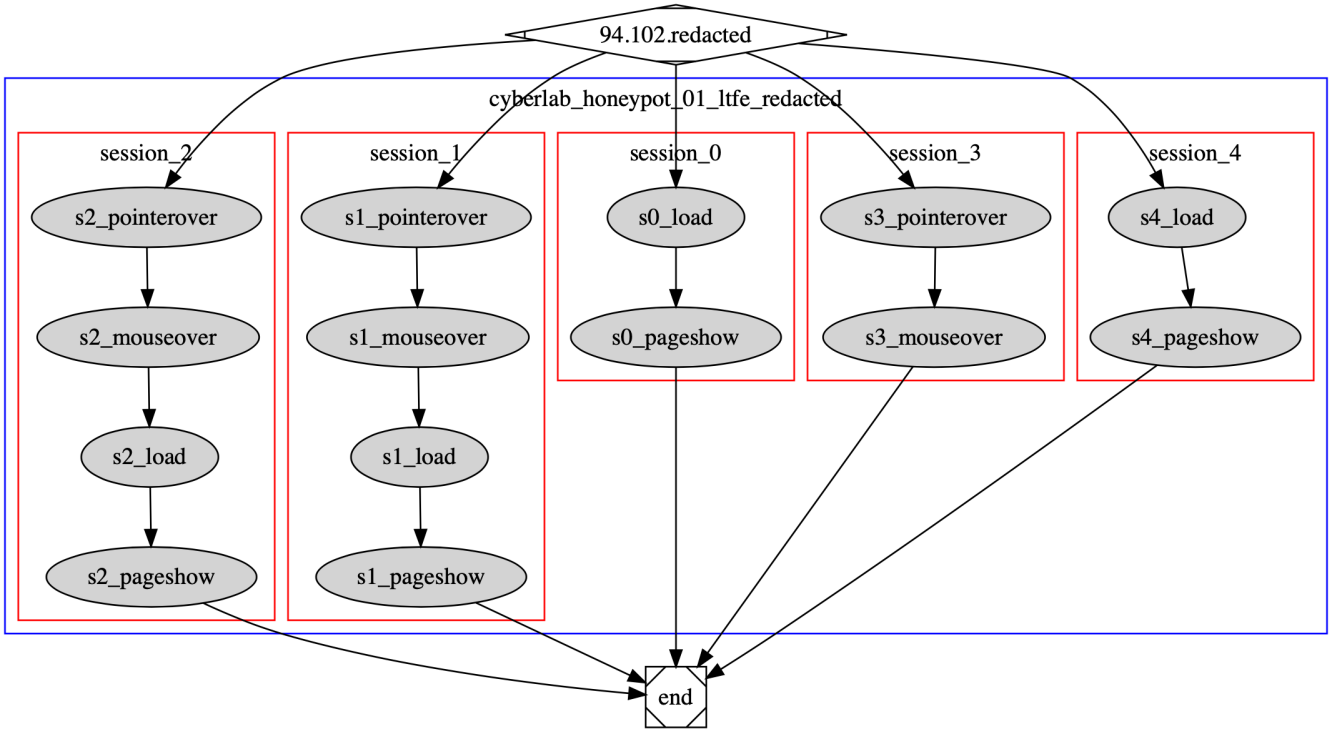


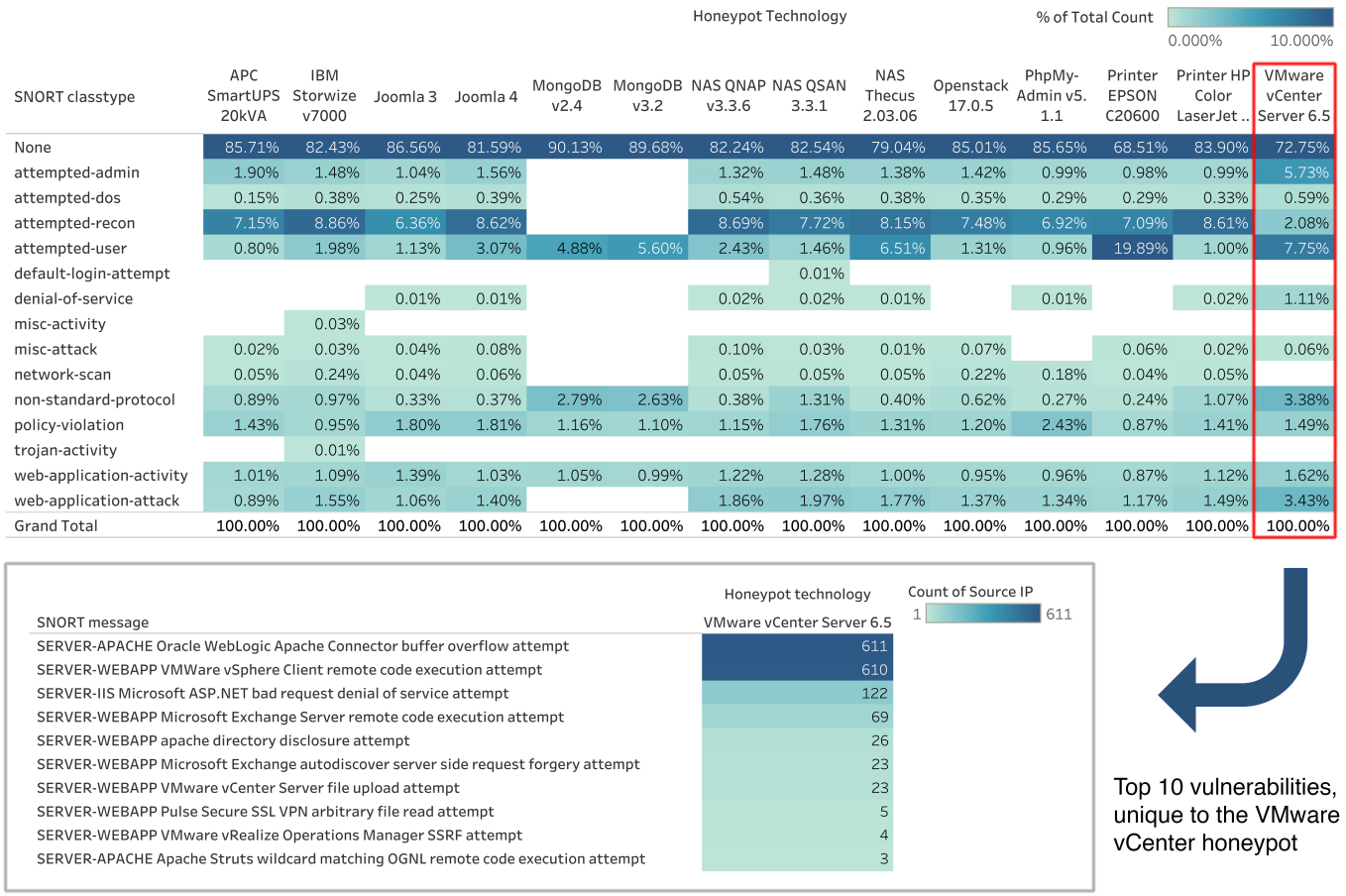
Figure 9. Visualization of the captured client-side event transitions for 5 individual sessions initiated by a known bot. The figure portrays relatively modest event sequences in comparison with the events and their transitions typically recorded during the user sessions.

3.5. Identification of Common Vulnerabilities and Exposures

To obtain a stronger signal of the attacker's activity and intent, we used the captured metadata to perform cross-validation with the CVE database and the rulesets of popular intrusion detection systems (IDSs). The CVE database includes unstructured text, and is thus difficult to query for automated lookups; on this front, we had some limited success with fine-tuning large language models, but we leave this approach for future work. On the other hand, rulesets of IDS systems such as SNORT are well-structured and intended for automation, which allowed us to cross-check every HTTP request against the entire collection, yielding multiple relevant CVEs. The results based on our experimental run are shown in Figure 10.

3.6. Experimental Attack Modeling and Visualization

Besides user and service modeling, we attempted to employ visual analytics in order to understand the user journey on the web application honeypots better and discover any potential referer tree inconsistencies indicating possible HTTP request replays with manually altered URL endpoints. The original idea behind this approach was an attempt at the detection of replayed HTTP requests originating from attack proxy tools, such as Burp Suite, where an attacker would modify the source request's parameters (e.g., destination URL to attempt directory traversal or file discovery) without removing its original referer header. In combination with reliable honeypot service models, such requests would violate the model's referer-to-URL mappings and break the valid referer tree, thus indicating a manual intervention during the proxied browsing session. We speculated this would be a potentially useful reconnaissance attack indicator, demonstrating the transition from passive web browsing to active, targeted discovery measures. Moreover, when combined with other attack signals (e.g., honeypot HTTP request properties, request history, and client-side user events) and their visualization, it could further provide visual clues to the attacker's reconnaissance journey and their transition to service exploitation. A prototype request flow visualization is shown in Figure 11.



SNORT classtype	APC SmartUPS 20kVA	IBM Storwize v7000	Joomla 3	Joomla 4	MongoDB v2.4	MongoDB v3.2	NAS QNAP v3.3.6	NAS QSAN 3.3.1	NAS Thecus 2.03.06	Openstack 17.0.5	PhpMy-Admin v5. 1.1	Printer EPSON C20600	Printer HP Color LaserJet ..	VMware vCenter Server 6.5
None	85.71%	82.43%	86.56%	81.59%	90.13%	89.68%	82.24%	82.54%	79.04%	85.01%	85.65%	68.51%	83.90%	72.75%
attempted-admin	1.90%	1.48%	1.04%	1.56%			1.32%	1.48%	1.38%	1.42%	0.99%	0.98%	0.99%	5.73%
attempted-dos	0.15%	0.38%	0.25%	0.39%			0.54%	0.36%	0.38%	0.35%	0.29%	0.29%	0.33%	0.59%
attempted-recon	7.15%	8.86%	6.36%	8.62%			8.69%	7.72%	8.15%	7.48%	6.92%	7.09%	8.61%	2.08%
attempted-user	0.80%	1.98%	1.13%	3.07%	4.88%	5.60%	2.43%	1.46%	6.51%	1.31%	0.96%	19.89%	1.00%	7.75%
default-login-attempt								0.01%						
denial-of-service			0.01%	0.01%			0.02%	0.02%	0.01%		0.01%		0.02%	1.11%
misc-activity		0.03%												
misc-attack	0.02%	0.03%	0.04%	0.08%			0.10%	0.03%	0.01%	0.07%		0.06%	0.02%	0.06%
network-scan	0.05%	0.24%	0.04%	0.06%			0.05%	0.05%	0.05%	0.22%	0.18%	0.04%	0.05%	
non-standard-protocol	0.89%	0.97%	0.33%	0.37%	2.79%	2.63%	0.38%	1.31%	0.40%	0.62%	0.27%	0.24%	1.07%	3.38%
policy-violation	1.43%	0.95%	1.80%	1.81%	1.16%	1.10%	1.15%	1.76%	1.31%	1.20%	2.43%	0.87%	1.41%	1.49%
trojan-activity		0.01%												
web-application-activity	1.01%	1.09%	1.39%	1.03%	1.05%	0.99%	1.22%	1.28%	1.00%	0.95%	0.96%	0.87%	1.12%	1.62%
web-application-attack	0.89%	1.55%	1.06%	1.40%			1.86%	1.97%	1.77%	1.37%	1.34%	1.17%	1.49%	3.43%
Grand Total	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

SNORT message	Honeypot technology VMware vCenter Server 6.5
SERVER-APACHE Oracle WebLogic Apache Connector buffer overflow attempt	611
SERVER-WEBAPP VMWare vSphere Client remote code execution attempt	610
SERVER-IIS Microsoft ASP.NET bad request denial of service attempt	122
SERVER-WEBAPP Microsoft Exchange Server remote code execution attempt	69
SERVER-WEBAPP apache directory disclosure attempt	26
SERVER-WEBAPP Microsoft Exchange autodiscover server side request forgery attempt	23
SERVER-WEBAPP VMware vCenter Server file upload attempt	23
SERVER-WEBAPP Pulse Secure SSL VPN arbitrary file read attempt	5
SERVER-WEBAPP VMware vRealize Operations Manager SSRF attempt	4
SERVER-APACHE Apache Struts wildcard matching OGNL remote code execution attempt	3

Figure 10. Identified vulnerability classes on the incoming requests. One of the services—VMware vCenter Server—is expanded in the bottom part of the figure to show the top 10 vulnerabilities that are unique just to this specific honeypot. Multiple VMware vCenter Server vulnerabilities are seen, such as a VMWare Sphere Client remote code execution attempt, VMware Center Server file upload attempt, and VMware Realize Operations Manager Server Side Request Forgery attempt.

However, contrary to the expected outcomes, our prototype implementation of request flow visualization did not result in the visualization of the prominent malicious actions but rather revealed the deficiencies in one of our implemented service models. The dubious attempts at referer-based threat hunting instead proved to be a surprisingly useful tool for visualizing (virtually) unlinked resources and invalid paths of the scraped web applications.

Figure 11 depicts a recorded session of HTTP requests during the initial access to one of the honeypot services. It portrays requests originating from the attacker's IP address and destined to the root path of the web service hosted at the honeypot's IP address. Subsequent requests then recursively use the response path as a referer when requesting new resources. However, due to an incomplete service model, the visualization system marked several paths as invalid or unavailable. Paths colored red indicate an illegal referer for the requested resource, while paths colored purple indicate an illegal target URL when using the current referer. As seen in the figure, an early service model would not foresee, for example, users simply refreshing the page. Of course, the honeypot would normally serve the refreshed website without issues; however, no semantic meaning regarding the user's action (i.e., page refresh) would be retained.

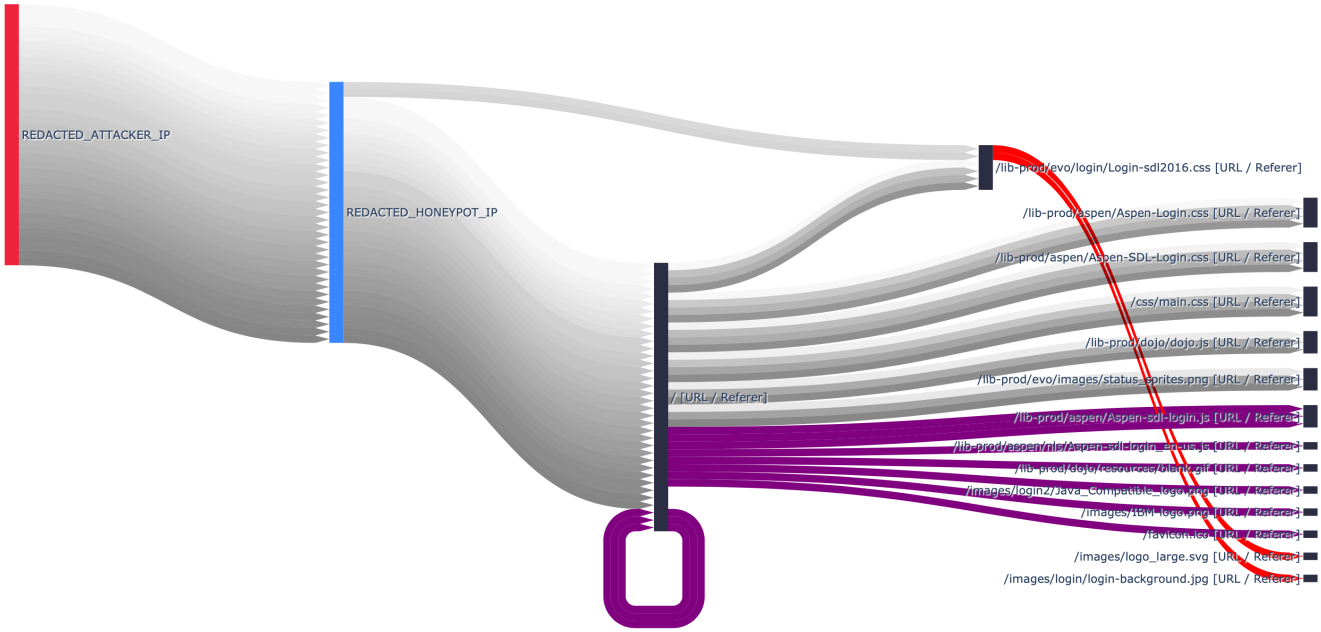


Figure 11. Experimental request flow visualization. The left side of the Sankey diagram depicts the attacker IP (red) originating HTTP requests to the honeypot server (cyan). Request URLs and referers are shown in dark blue. Traffic flows from left to right, and time flow for grouped requests is portrayed from top to bottom, where link opacity (ranging from transparent to dark gray) indicates their relative timestamp. Individual links indicate specific requests originating at the start node (IP address, referer) and terminating in the target node (target IP, URL). Requests colored in red and purple depict the semantic service model's deficiencies, indicating an illegal referer for access to the requested resource, or an illegal target URL using the current referer, respectively.

Nevertheless, upon refinement of the semantic service models, the proposed visualization was able to convey arbitrary link features within the visualized request tree. This is shown in Figure 12, visualizing a complex user journey through a web application and disclosing two POST requests (colored red). At the top left, the figure reveals the user's true origin as FOFA, a cyberspace-mapping search engine that likely indexed our honeypot. The user then interacted relatively heavily with the printer honeypot, including by visiting numerous configuration subpages, before ultimately attempting to alter the protected storage settings and resetting the device to factory defaults (requests colored red). While the shown experimental visualization is capable of conveying such features, its usefulness remains very limited due to the high visual complexity, lacking representations of timing data, and significant overplotting.

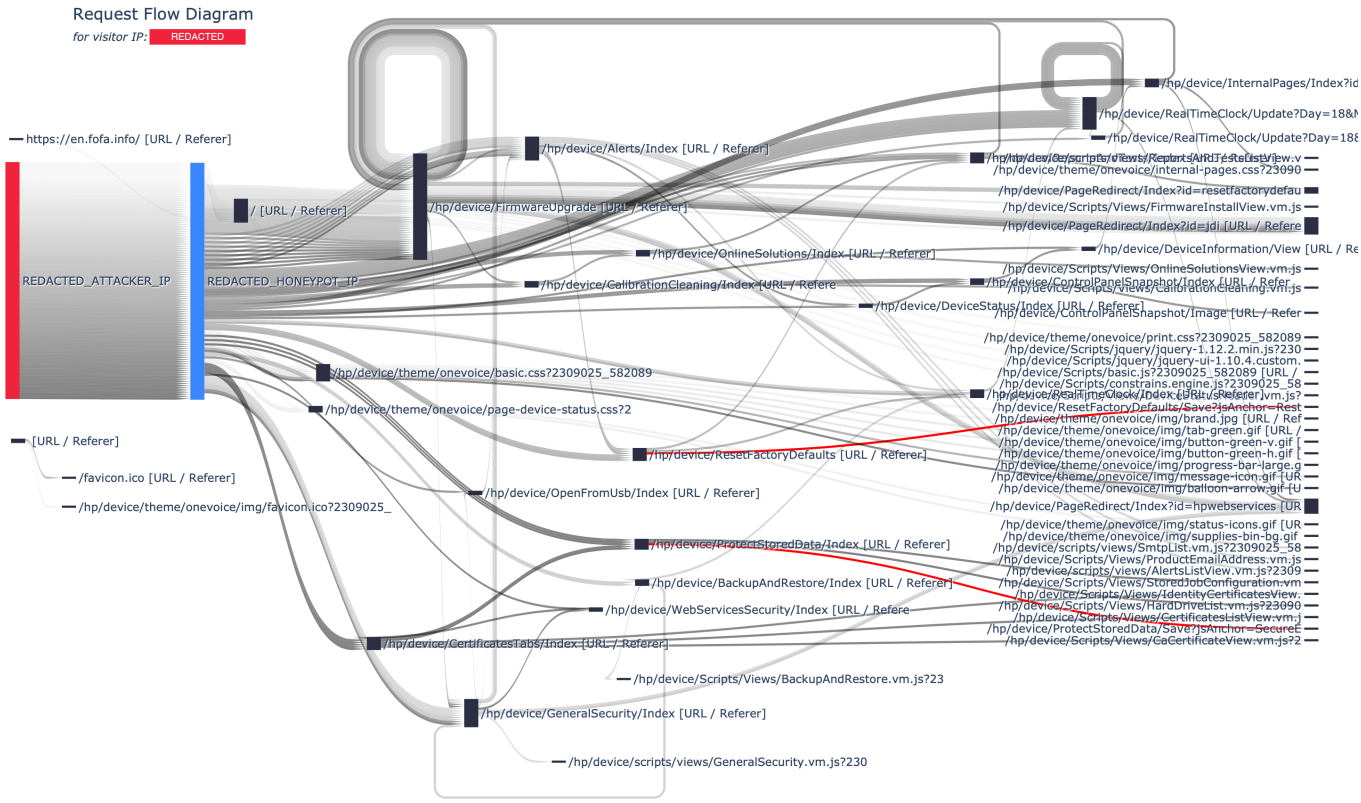


Figure 12. Experimental request flow visualization depicting relatively rich interactions with a honeypot. The referer at the top left side of the diagram reveals the visitor's true origin (FOFA cyberspace-mapping search engine). Request URLs and referers are shown in dark blue. Traffic flows from left to right, and time flow for grouped requests is portrayed from top to bottom, where link opacity (ranging from transparent to dark gray) indicates their relative timestamp. Individual links indicate specific requests originating at the start node (IP address, referer) and terminating in the target node (target IP, URL). Requests colored red reveal two POST requests, quickly revealing the attacker's attempt to alter the protected storage settings and factory reset the emulated printer.

4. Discussion

During the design and development of our solution, as well as throughout the data collection, processing, and analysis, we faced several design decisions and challenges. We tested various system configurations, revised the architecture design, and iteratively improved our honeypot service models to build a robust attack monitoring platform to conduct the experiment. Therefore, we now briefly discuss some of our findings, lessons learned, and challenges encountered.

The implementation of HTTP honeypots based on crawling and caching of web services has proven to be difficult. Despite the development of an automated scraping solution, numerous challenges prevented us from automating the deployment process. Scraped web services required manual verification of their functionality. Many services also included dynamic content, requiring manual field examination and implementation of dynamic variables to replicate the original application convincingly. Difficult service modeling also limited the robustness of our early honeypot implementations, requiring their iterative improvements. Luckily, we were able to implement these improvements effectively due to the well-designed, centralized logging system and by using the request flow visualizations of real visitor data to identify spotty service coverage and problematic client-side caching policies. Moreover, we also crawled error code pages (e.g., 404, 500, etc.) where encountered; however, it is hard to know where these codes are actually warranted without referencing a real device. For these reasons, we plan on reserving the

crawl-and-store caching mechanism for only the simplest systems requiring a minimal level of interaction. For all other use cases, transparent pass-through proxying should be used to forward the requests to either real, virtualized, or emulated devices and services.

Related to the selection of supported services, it is clear that a methodology to evaluate and rank candidate services systematically is needed. However, the presented threat impact assessment formula proved to be difficult to estimate in practice and thus has to be approximated with various heuristics, assumptions, and rules of thumb. For this reason, one of the original platform requirements called for the honeypots to select the emulated technology dynamically from a larger pool of supported services and then lock the responses to the particular source IP address making the contact. In this manner, a single IP address could appear as multiple devices to multiple attackers at the same time, and a greater number of dynamic responses could be served and tested for attractiveness, in a sense empirically exploring the space of available services. Similarly, another goal of the early platform implementation was an adjustable ratio between such exploration (testing new response variations) and exploitation (obtaining more relevant attacker data). However, after several months of operation in this mode, we aborted the experiment to increase the realism of honeypots and make them respond in a fixed manner, so the attacker could return to the service after an arbitrary amount of time and encounter the same technology.

5. Conclusions

In this work, we presented an approach to the design, development, deployment, and evaluation of a flexible and scalable HTTP honeypot platform. We introduced IoT and HTTP honeypots and argued for their use in the detection and analysis of novel and emerging cyber threats. We reviewed the current state-of-the-art research on honeypot solutions and identified their applications, advantages, and drawbacks. We then described the motivation for the design of a scalable honeypot platform with the ability to emulate a wide range of IoT devices, cloud services, and web applications. We identified the necessary requirements for such a system and proposed a distributed platform architecture with centralized management and logging. Next, we detailed the technical implementation of the platform and set up a four-month-long experiment for system validation. Lastly, we described the captured honeypot data and presented our findings on attack interception, bot detection, exploit identification, and attacker, user, and service modeling.

Despite many of the encountered challenges, we managed to develop a robust honeypot platform, adhering to all of the priority 1 and priority 2 initial identified requirements. The platform has proven to be an invaluable source of data, enabling first-hand observations of the dynamic cybersecurity landscape. Its distributed nature, extensibility, and scalability, along with its integration with multiple third-party databases and threat intelligence feeds, support a global honeynet operation with a centralized data storage mechanism for rapid detection and analysis of potential novel threats. The implemented server-side and client-based approaches to data collection for user and attacker modeling have proven valuable for the detection of potential threat actors and promise their further categorization amongst bots, scanners, crawlers, and advanced persistent threats. Browser fingerprinting yielded a wealth of data useful for multiple purposes, including the potential for prediction of attack escalation risks.

During the course of the conducted experiment, we used the platform to capture over 188,287 HTTP requests from 17,176 unique IP addresses targeting the 14 implemented honeypot services. We proposed a novel approach toward internet scanner identification and classified 3160 IP addresses belonging to various search engines. We identified 5006 IP addresses originating malicious traffic from 1327 ASNs. We demonstrated the successful capture of automated targeted attacks using service-specific exploits against particular honeypot services and matched them with CVE entries. Furthermore, we identified 159 unique IP addresses that interpreted the honeypot-delivered JavaScript profiling code. Based on the latter, we captured 26,502 client-side events and proposed a rudimentary mechanism for

the classification of browser automation tools and human visitors. We identified 15 unique IP addresses with human actors performing web service exploration. We examined these visitor sessions and demonstrated the detection of malicious intent in the form of actions against honeypots that would incur data loss or denial of service on real devices. However, we have not yet observed any human actors escalating the attack from active web reconnaissance to exploitation using known vulnerabilities in the CVE database. Although the dataset of such user sessions is currently still modest, it presents actual telemetry from attackers, by definition. Conversely, the collected metadata on incoming requests from network scanners are extensive and currently one of the most useful results of the platform, as they constitute a rich database that is updated in an automated manner.

Within our future work, we plan on improving the mechanisms for matching the collected data with the CVE database entries and various cyber threat intelligence feeds, as we have identified that the current implementation can produce inconsistent results and has low specificity (i.e., a high rate of false positives). We plan to expand the deployed honeynet into a global, geographically dispersed HTTP sensor network to maximize the attack surface area, capture larger datasets, and compare attacker tactics, techniques, and procedures between the target node networks. Additionally, we plan to test further data integrations, such as with the MITRE ATT&CK framework and other feeds of IDS and WAF rules. One of the very promising venues of research in this field is also the extraction of prominent attack features using machine learning approaches, including leveraging the fine-tuned large language models on the collected datasets.

Author Contributions: Conceptualization, U.S. and M.R.; methodology, L.Š.J., M.K., M.R. and U.S.; software, L.Š.J., M.R. and M.K.; validation, L.Š.J. and M.R.; formal analysis, M.K.; investigation, M.R., L.Š.J., M.V. and U.S.; resources, A.K., M.V. and U.S.; data curation, L.Š.J., M.R. and U.S.; writing—original draft preparation, M.R., M.V. and U.S.; writing—review and editing, M.R., M.V., A.K. and U.S.; visualization, M.R. and U.S.; supervision, U.S.; project administration, U.S.; funding acquisition, U.S., M.V. and A.K. All authors have read and agreed to the published version of the manuscript.

Funding: The research was supported by the Slovenian Research and Innovation Agency and Government Information Security Office of Slovenia through the research project "Exposure of Modern Information and Communication Infrastructures to Cyberattacks" (Grant number V2-2125) and the research program "Decentralized Solutions for the Digitalization of Industry and Smart Cities and Communities" (Grant number P2-0425). The APC was funded by the Slovenian Research and Innovation Agency, grant number P2-0425.

Data Availability Statement: The data presented in this study are available upon request to the corresponding author. The data in raw form are not publicly available due to their sensitive nature and privacy restrictions.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial intelligence
ALG	Application Layer Gateway
API	Application programming interface
AS	Autonomous system
ASN	Autonomous System number
CMS	Content management system
COTS	Commercial off-the-shelf
CPU	Central processing unit
CSS	Cascading Style Sheets

CTI	Cyber threat intelligence
CVE	Common Vulnerabilities and Exposures
DB	Database
DNS	Domain name system
GPU	Graphics processing unit
GUI	Graphical user interface
HTML	HyperText Markup Language
HTTPS	Hypertext Transfer Protocol Secure
ICS	Industrial control system
IDS	Intrusion detection system
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion prevention system
ISP	Internet service provider
IT	Information technology
JS	JavaScript
LLM	Large language model
ML	Machine learning
NAS	Network attached storage
OS	Operating system
OT	Operational technology
PPDR	Public protection and disaster relief
REST	Representational state transfer
RTSP	Real-Time Streaming Protocol
SQL	Structured query language
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UA	User agent
UDP	User Datagram Protocol
UPS	Uninterruptible power supply
URL	Uniform Resource Locator
VANET	Vehicular ad hoc networks
VM	Virtual machine
VPN	Virtual private network
WAF	Web application firewall
XSS	Cross-site scripting

References

1. Al-Garadi, M.A.; Mohamed, A.; Al-Ali, A.K.; Du, X.; Ali, I.; Guizani, M. A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1646–1685. [CrossRef]
2. Franco, J.; Aris, A.; Canberk, B.; Uluagac, A.S. A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2351–2383. [CrossRef]
3. Makhdoom, I.; Abolhasan, M.; Lipman, J.; Liu, R.P.; Ni, W. Anatomy of Threats to the Internet of Things. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1636–1675. [CrossRef]
4. Matheu, S.N.; Hernández-Ramos, J.L.; Skarmeta, A.F.; Baldini, G. A Survey of Cybersecurity Certification for the Internet of Things. *ACM Comput. Surv.* **2021**, *53*, 115. [CrossRef]
5. Dionaea. Available online: https://github.com/DinoTools/dionaea (accessed on 21 July 2023).
6. Cowrie. Available online: https://github.com/cowrie/cowrie (accessed on 28 July 2023).
7. Honeytrap. Available online: https://github.com/honeytrap/honeytrap (accessed on 28 July 2023).
8. Kuskov, V.; Kuzin, M.; Shmelev, Y.; Makrushin, D.; Grachev, I. Honeypots and the Internet of Things. Available online: https://securelist.com/honeypots-and-the-internet-of-things/78751/ (accessed on 21 July 2023).
9. Surber, J.G.; Zantua, M. Intelligent Interaction Honeypots for Threat Hunting within the Internet of Things. *J. Colloq. Inf. Syst. Secur. Educ.* **2022**, *9*, 5. [CrossRef]
10. Metongnon, L.; Sadre, R. Beyond Telnet: Prevalence of IoT Protocols in Telescope and Honeypot Measurements. In Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity; WTMC'18, Budapest, Hungary, 20 August 2018; Association for Computing Machinery: New York, NY, USA; pp. 21–26. [CrossRef]

11. Semic, H.; Mrdovic, S. IoT Honeypot: A Multi-Component Solution for Handling Manual and Mirai-based Attacks. In Proceedings of the 2017 25th Telecommunication Forum (TELFOR), Belgrade, Serbia, 21–22 November 2017; pp. 1–4. [CrossRef]
12. Manzanares, A.G. *The Construction of a Virtual, Low-Interaction IoT Honeypot*; Semantic Scholar: Seattle, WA, USA, 2017.
13. Dowling, S.; Schukat, M.; Melvin, H. Data-Centric Framework for Adaptive Smart City Honeynets. In Proceedings of the 2017 Smart City Symposium Prague (SCSP), Prague, Czech Republic, 25–26 May 2017; pp. 1–7. [CrossRef]
14. Chen, D.D.; Egele, M.; Woo, M.; Brumley, D. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In Proceedings of the Proceedings 2016 Network and Distributed System Security Symposium; Internet Society, San Diego, CA, USA, 21–24 February 2016. [CrossRef]
15. Wang, M.; Santillan, J.; Kuipers, F. ThingPot: An Interactive Internet-of-Things Honeypot. Available online: http://arxiv.org/abs/1807.04114 (accessed on 29 July 2023).
16. Vishwakarma, R.; Jain, A.K. A Honeypot with Machine Learning Based Detection Framework for Defending IoT Based Botnet DDoS Attacks. In Proceedings of the 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 23–25 April 2019; pp. 1019–1024. [CrossRef]
17. Luo, T.; Xu, Z.; Jin, X.; Jia, Y.; Ouyang, X. IoTCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices. *Black Hat* **2017**, *1*, 1–11.
18. Zhou, Y. Chameleon: Towards adaptive honeypot for internet of things. In Proceedings of the ACM Turing Celebration Conference—China, Chengdu, China, 17–19 May 2019. [CrossRef]
19. Vetterl, A.; Clayton, R. Honware: A Virtual Honeypot Framework for Capturing CPE and IoT Zero Days. In Proceedings of the 2019 APWG Symposium on Electronic Crime Research (eCrime), Pittsburgh, PA, USA, 13–15 November 2019; pp. 1–13. [CrossRef]
20. Guarnizo, J.; Tambe, A.; Bhunia, S.S.; Ochoa, M.; Tippenhauer, N.; Shabtai, A.; Elovici, Y. SIPHON: Towards Scalable High-Interaction Physical Honeypots. *arXiv* **2017**. arXiv:1701.02446.
21. Shrivastava, R.K.; Bashir, B.; Hota, C. Attack Detection and Forensics Using Honeypot in IoT Environment. In Proceedings of the Distributed Computing and Internet Technology, Bhubaneswar, India, 10–13 January 2019; Fahrnberger, G., Gopinathan, S., Parida, L., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2019; pp. 402–409. [CrossRef]
22. Pauna, A.; Bica, I.; Pop, F.; Castiglione, A. On the Rewards of Self-Adaptive IoT Honeypots. *Ann. Telecommun.* **2019**, *74*, 501–515. [CrossRef]
23. Lingenfelter, B.; Vakilinia, I.; Sengupta, S. Analyzing Variation Among IoT Botnets Using Medium Interaction Honeypots. In Proceedings of the 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 6–8 January 2020; pp. 761–767. [CrossRef]
24. Wang, B.; Dou, Y.; Sang, Y.; Zhang, Y.; Huang, J. IoTCMal: Towards A Hybrid IoT Honeypot for Capturing and Analyzing Malware. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–7. [CrossRef]
25. Hakim, M.A.; Aksu, H.; Uluagac, A.S.; Akkaya, K. U-PoT: A Honeypot Framework for UPnP-Based IoT Devices. In Proceedings of the 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC), Orlando, FL, USA, 17–19 November 2018; pp. 1–8. [CrossRef]
26. Trajanovski, T.; Zhang, N. An Automated and Comprehensive Framework for IoT Botnet Detection and Analysis (IoT-BDA). *IEEE Access* **2021**, *9*, 124360–124383. [CrossRef]
27. Krishna, R.R.; Priyadarshini, A.; Jha, A.V.; Appasani, B.; Srinivasulu, A.; Bizon, N. State-of-the-Art Review on IoT Threats and Attacks: Taxonomy, Challenges and Solutions. *Sustainability* **2021**, *13*, 9463. [CrossRef]
28. Pa, Y.M.; Suzuki, S.; Yoshioka, K.; Matsumoto, T.; Kasama, T.; Rossow, C. IoTPOT: A Novel Honeypot for Revealing Current IoT Threats. *J. Inf. Process.* **2016**, *24*, 522–533. [CrossRef]
29. Baş Seyyar, M.; Çatak, F.Ö.; Gül, E. Detection of Attack-Targeted Scans from the Apache HTTP Server Access Logs. *Appl. Comput. Inform.* **2018**, *14*, 28–36. [CrossRef]
30. Mx, P. HoneyPy. Available online: https://github.com/foospidy/HoneyPy (accessed on 30 July 2023).
31. Pa, Y.M.P.; Suzuki, S.; Yoshioka, K.; Matsumoto, T.; Kasama, T.; Rossow, C. IoTPOT: Analysing the Rise of IoT Compromises. In Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT 15), Washington, DC, USA, 10–11 August 2015.
32. Dowling, S.; Schukat, M.; Melvin, H. A ZigBee Honeypot to Assess IoT Cyberattack Behaviour. In Proceedings of the 2017 28th Irish Signals and Systems Conference (ISSC), Killarney, Ireland, 20–21 June 2017; pp. 1–6. [CrossRef]
33. Tambe, A.; Aung, Y.L.; Sridharan, R.; Ochoa, M.; Tippenhauer, N.O.; Shabtai, A.; Elovici, Y. Detection of Threats to IoT Devices Using Scalable VPN-forwarded Honeypots. In Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, Dallas, TX, USA, 25–27 March 2019. [CrossRef]
34. Ramakrishnan, K.; Gokul, P.; Nigam, R. Pandora: An IOT Based Intrusion Detection Honeypot with Real-time Monitoring. In Proceedings of the 2021 International Conference on Forensics, Analytics, Big Data, Security (FABS), Bengaluru, India, 21–22 December 2021; Volume 1, pp. 1–7. [CrossRef]
35. Bartwal, U.; Mukhopadhyay, S.; Negi, R.; Shukla, S. Security Orchestration, Automation, and Response Engine for Deployment of Behavioural Honeypots. In Proceedings of the 2022 IEEE Conference on Dependable and Secure Computing (DSC), Edinburgh, UK, 22–24 June 2022; pp. 1–8. [CrossRef]

36. Kato, S.; Tanabe, R.; Yoshioka, K.; Matsumoto, T. Adaptive Observation of Emerging Cyber Attacks Targeting Various IoT Devices. In Proceedings of the 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), Bordeaux, France, 17–21 May 2021. Available online: https://ieeexplore.ieee.org/document/9464004 (accessed on 30 July 2023).
37. Tabari, A.Z.; Ou, X. A First Step Towards Understanding Real-world Attacks on IoT Devices. *arXiv* **2020**, arXiv:2003.01218.
38. ENISA Threat Landscape Report 2018. Available online: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018 (accessed on 29 July 2023).
39. Sedlar, U. Network Telescope: Insights from a Decade of Observations. *Electrotech. Rev. Vestn.* **2022**, *89*, 198–204.
40. Sedlar, U.; Južnič, L.Š.; Volk, M. An Iteratively-Improving Internet-of-Things Honeypot Experiment. In Proceedings of the 2020 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom), Graz, Austria, 7–9 July 2020; pp. 1–6. [CrossRef]
41. Bauer, J.M.; van Eeten, M.J.G. Cybersecurity: Stakeholder Incentives, Externalities, and Policy Options. *Telecommun. Policy* **2009**, *33*, 706–719. [CrossRef]
42. CONCORDIA: Cyber Security Competence for Research and Innovation. In *Work Package 4: Policy and the European Dimension, Deliverable D4.3: 3rd Year Report on Cybersecurity Threats*; Technical Report; European Commission: Luxembourg, 2020.
43. Fischer-Hübner, S.; Alcaraz, C.; Ferreira, A.; Fernandez-Gago, C.; Lopez, J.; Markatos, E.; Islami, L.; Akil, M. Stakeholder Perspectives and Requirements on Cybersecurity in Europe. *J. Inf. Secur. Appl.* **2021**, *61*, 102916. [CrossRef]
44. Kren, M.; Kos, A.; Sedlar, U. Estimating Application Cyberthreat Impact Score for Honeypot Coverage Prioritization. In Proceedings of the 2022 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom), Graz, Austria, 12–14 July 2022; pp. 1–6. [CrossRef]
45. JVN iPedia. Available online: https://jvndb.jvn.jp/en/ (accessed on 7 July 2023).
46. Selenium. Available online: https://www.selenium.dev/ (accessed on 31 July 2023).
47. Puppeteer. Available online: https://pptr.dev/ (accessed on 31 July 2023).
48. Playwright. Available online: https://playwright.dev/python/ (accessed on 31 July 2023).